L 12

```
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS
SSS              YYY         YYY  SSS
SSS              YYY         YYY  SSS
SSS              YYY         YYY  SSS
SSS                 YYY   YYY     SSS
SSS                 YYY   YYY     SSS
SSS                 YYY   YYY     SSS
   SSSSSSSSS           YYY           SSSSSSSSS
   SSSSSSSSS           YYY           SSSSSSSSS
   SSSSSSSSS           YYY           SSSSSSSSS
            SSS        YYY                    SSS
            SSS        YYY                    SSS
            SSS        YYY                    SSS
            SSS        YYY                    SSS
            SSS        YYY                    SSS
            SSS        YYY                    SSS
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS
```

```
_$

Ps
--

YZ

Z$

Z$

Z$


Z$


Z$




Z$

Z$

Z$

Z$

Z$

Z$


Z$
```

```
**FILE**ID**IOSUBNPAG                                  K  2




  IIIIII      OOOOOO      SSSSSSSS  UU      UU  BBBBBBBB    NN      NN  PPPPPPPP      AAAAAA      GGGGGGGG
  IIIIII      OOOOOO      SSSSSSSS  UU      UU  BBBBBBBB    NN      NN  PPPPPPPP      AAAAAA      GGGGGGGG
    II      OO      OO  SS          UU      UU  BB      BB  NN      NN  PP      PP  AA      AA  GG
    II      OO      OO  SS          UU      UU  BB      BB  NN      NN  PP      PP  AA      AA  GG
    II      OO      OO  SS          UU      UU  BB      BB  NNNN    NN  PP      PP  AA      AA  GG
    II      OO      OO  SS          UU      UU  BB      BB  NNNN    NN  PP      PP  AA      AA  GG
    II      OO      OO    SSSSSS    UU      UU  BBBBBBBB    NN  NN  NN  PPPPPPPP    AA      AA  GG
    II      OO      OO    SSSSSS    UU      UU  BBBBBBBB    NN  NN  NN  PPPPPPPP    AA      AA  GG
    II      OO      OO          SS  UU      UU  BB      BB  NN    NNNN  PP          AAAAAAAAAA  GG  GGGGGG
    II      OO      OO          SS  UU      UU  BB      BB  NN    NNNN  PP          AAAAAAAAAA  GG  GGGGGG
    II      OO      OO          SS  UU      UU  BB      BB  NN      NN  PP          AA      AA  GG      GG      ....
    II      OO      OO          SS  UU      UU  BB      BB  NN      NN  PP          AA      AA  GG      GG      ....
  IIIIII      OOOOOO    SSSSSSSS    UUUUUUUUUU  BBBBBBBB    NN      NN  PP          AA      AA    GGGGGG        ....
  IIIIII      OOOOOO    SSSSSSSS    UUUUUUUUUU  BBBBBBBB    NN      NN  PP          AA      AA    GGGGGG        ....


LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

Table of contents

```
   (3)       239        CANCEL I/O ON CHANNEL
   (4)       273        Handle Last Channel Deassign
   (5)       332        FILL DIAGNOSTIC BUFFER
   (6)       363        RELEASE I/O CHANNEL
   (7)       418        REQUEST I/O CHANNEL
   (8)       478        I/O Request Completion Processing for Class Drivers
   (9)       526        I/O REQUEST COMPLETION PROCESSING
   (10)      637        MOUNT VERIFICATION HELPER
   (11)      670        INITIATE I/O FUNCTION ON DEVICE
   (12)      708        Allocate Buffered Data Path
   (14)      819        Release Buffered Data Path
   (15)      904        REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
   (16)      945        REQUEST UNIBUS MAP REGISTERS
   (17)      980        ALLOCATE UNIBUS MAP REGISTERS
   (18)     1087        Allocate a specific set of UNIBUS Map Registers
   (19)     1196        Permanently Allocate UNIBUS Map Registers
   (21)     1321        Release UNIBUS Map Registers
   (23)     1529        RETURN TO CALLER
   (24)     1548        WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
   (25)     1582        WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
   (26)     1618        ALLOCATE SYSTEM PAGE TABLE
   (27)     1653        CONVERT DEVICE NAME AND UNIT
   (28)     1934        BROADCAST TO A TERMINAL
   (29)     2047        BROADCAST EMERGENCY MESSAGE TO CONSOLE
   (30)     2131        SCAN THE I/O DATA BASE
   (31)     2191        SCAN THE I/O DATA BASE BOTH PRIMARY & SECONDARY PATHS
   (32)     2262        IOC$CTRLINIT - Call driver controller init. routine
   (33)     2327        IOC$UNITINIT - Call driver unit init. routine
   (34)     2406        Parse Device Name String
   (35)     2587        Search I/O Database for Device
   (36)     2751        Continue I/O Database Search
   (37)     2801        Check UCB Against Search Rules
   (38)     2901        IOC$THREADCRB
```

```
0000            .TITLE  IOSUBNPAG - NONPAGED I/O RELATED SUBROUTINES
0000            .IDENT  'V04-000'
0000    ;****************************************************************************
0000    ;*                                                                          *
0000    ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000    ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000    ;*  ALL RIGHTS RESERVED.                                                    *
0000    ;*                                                                          *
0000    ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
0000    ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
0000    ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
0000    ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
0000    ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
0000    ;*  TRANSFERRED.                                                            *
0000    ;*                                                                          *
0000    ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
0000    ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
0000    ;*  CORPORATION.                                                            *
0000    ;*                                                                          *
0000    ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
0000    ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    ;*                                                                          *
0000    ;*                                                                          *
0000    ;****************************************************************************
0000    ;
0000    ; D. N. CUTLER 13-JUN-76
0000    ;
0000    ;
0000    ; NONPAGED I/O RELATED SUBROUTINES
0000    ;
0000    ; MODIFIED BY:
0000    ;
0000    ;       V03-038 WMC0004         Wayne Cardoza           23-Aug-1984
0000    ;               Add routine for emergency message to console.
0000    ;
0000    ;       V03-037 WMC0003         Wayne Cardoza           14-Aug-1984
0000    ;               Fix ROW0409 to restore the correct register.
0000    ;
0000    ;       V03-036 ACG0442         Andrew C. Goldstein,    7-Aug-1984  17:52
0000    ;               Save R8 in IOC$LAST_CHAN; fix order of tests in IOC$TESTUNIT
0000    ;               for correct allocation and mount checks. Fix handling of
0000    ;               lock value block on device lock in IOC$TESTUNIT.
0000    ;
0000    ;       V03-035 ROW0409         Ralph O. Weber          6-AUG-1984
0000    ;               Fix release map registers processing of requests waiting for
0000    ;               map registers.  Restore saved fork registers -- including the
0000    ;               PDT address -- before the calling IOC$ALOMAPUDA at
0000    ;               REALLOC_CD_MAPREGS.
0000    ;
0000    ;       V03-034 TCM0006         Trudy C. Matthews       20-Jul-1984
0000    ;               Add routine IOC$THREADCRB.
0000    ;
0000    ;       V03-033 WMC0002         Wayne Cardoza           03-May-1984
0000    ;               Add support for MNTVERPND bit.
0000    ;
0000    ;       V03-032 RAS0300         Ron Schaefer            2-May-1984
0000    ;               Change IOC$CVT_DEVNAM to only prefix cluster node names if
```

```
0000  ;                the DEV$V_NNM device characteristic is set in UCB$L_DEVCHAR2.
0000  ;                Add additional itemcode (4) to IOC$CVT_DEVNAM to provide
0000  ;                the device name string sans unit number.
0000  ;
0000  ;        V03-031 TMK0001         Todd M. Katz            23-Apr-1984
0000  ;                Remove the $LOGDEF data definitions.
0000  ;
0000  ;        V03-030 RLRPDTADP       Robert L. Rappaport     9-Apr-1984
0000  ;                Modify entrypoints used for allocating and deallocating
0000  ;                Buffered Data Paths and UNIBUS Map Registers for UQPORTS (UDA),
0000  ;                to pickup pointer for ADP from PDT$L_ADP(R4).
0000  ;
0000  ;        V03-029 ACG0414         Andrew C. Goldstein,    30-Mar-1984  15:49
0000  ;                Minor parse and searching fixes in IOC$SEARCH...
0000  ;                add IOC$V_ALLOC to force allocation
0000  ;
0000  ;        V03-028 ACG0406         Andrew C. Goldstein,    16-Mar-1984  15:42
0000  ;                Fix bugs in searching for allocation class
0000  ;
0000  ;        V03-027 ACG0399         Andrew C. Goldstein,    24-Feb-1984  22:28
0000  ;                Add IOC$LAST_CHAN subroutine, and move in internal I/O
0000  ;                database parse and search routines, so they can be called
0000  ;                by IPC.
0000  ;
0000  ;        V03-026 RLRMAPSP        Robert L. Rappaport     15-Feb-1984
0000  ;                Correct bug in BEQL destination in IOC$ALOUBAMAPSP that is
0000  ;                only triggered if the range specified, coincides with the
0000  ;                exact end of an extent of map registers.
0000  ;
0000  ;        V03-025 ROW0292         Ralph O. Weber          4-FEB-1984
0000  ;                Fix branch displacements broken by movement of EXE$MOUNTVER to
0000  ;                SYSLOAxxx.
0000  ;
0000  ;        V03-024 KPL0001         Peter Lieberwirth       7-Nov-1983
0000  ;                Add paths for new processors to CPUDISP invocation.
0000  ;
0000  ;        V03-023 ROW0244         Ralph O. Weber          17-OCT-1983
0000  ;                Change the IOC$CVT_DEVNAM name string formation rules to
0000  ;                eliminate _$1$TTA0: and other allocation class based names
0000  ;                for devices which can never be dual pathed.  See routine
0000  ;                comments for details of current operation mode.
0000  ;
0000  ;        V03-022 ROW0239         Ralph O. Weber          11-OCT-1983
0000  ;                Fix IOC$CVT_DEVNAM to not insert node name or trailing dollar
0000  ;                sign when node name is null.  Also correct comments describing
0000  ;                the R4 argument to IOC$CVT_DEVNAM.
0000  ;
0000  ;        V03-021 ROW0234         Ralph O. Weber          5-OCT-1983
0000  ;                Change IOC$CVT_DEVNAM to produce $allocation-class$device
0000  ;                strings completely in ASCII, when allocation class output is
0000  ;                requested.  In the process rip up the whole thing because that
0000  ;                was the only way to get something that worked and didn't
0000  ;                occupy all non-page memory
0000  ;
0000  ;        V03-020 TCM0005         Trudy C. Matthews       5-OCT-1983
0000  ;                Add IOC$SCAN_IODB_2P which is functionally the same as
0000  ;                IOC$SCAN_IOCB except that both primary and secondary paths to
```

```
0000 ;              a device are scanned.
0000 ;
0000 ;      V03-019 KDM0084         Kathleen D. Morse       26-Sep-1983
0000 ;              Added MicroVAX I support to CPUDISP macros.
0000 ;
0000 ;      V03-018 ROW0221         Ralph O. Weber          8-SEP-1983
0000 ;              Change IOC$UNITINIT to look for a unit initialization routine
0000 ;              in the DDT before looking in the CRB.  See the note in the
0000 ;              routine's header for details.
0000 ;
0000 ;      V03-017 ROW0203         Ralph O. Weber          5-AUG-1983
0000 ;              Add two new routines IOC$CTRLINIT and IOC$UNITINIT.  These are
0000 ;              the proscribed mechanism for calling device driver controller
0000 ;              and unit initialization routines.  These routines correctly
0000 ;              setup for, locate, and call the appropriate driver routines.
0000 ;
0000 ;      V03-016 TCM0004         Trudy C. Matthews       26-Jul-1983
0000 ;              Change IOC$CVT_DEVNAM to return the <allocation_class>+
0000 ;              <devnam> form of device name if R4 > 0.
0000 ;
0000 ;      V03-015 RLRBYTEOFF      Robert L. Rappaport     27-Jun-1983
0000 ;              Correct error in IOC$REQDATAPUDA.  Error is that this
0000 ;              routine has operated in a NOWAIT mode, that is, if no
0000 ;              Buffered Datapath was available, we just used the
0000 ;              Direct Datapath.  Unfortunately, this doesn't work on
0000 ;              780's and 790's if the user buffer is located at an
0000 ;              odd byte address since Byte Offset doesn't work on the
0000 ;              Direct Datapath for the UNIBUS Adapters on these
0000 ;              processors.
0000 ;
0000 ;      V03-014 LMPBUILD        L. Mark Pilant,         26-Jun-1983  23:11
0000 ;              Change references from TTY$K_WB_HDRLEN to TTY$K_WB_LENGTH.
0000 ;
0000 ;      V03-013 TCM0003         Trudy C. Matthews       17-Jun-1983
0000 ;              Change the way cluster-style device names are conditionally
0000 ;              returned, such that cluster-style names are returned for
0000 ;              local disk devices if the system is participating in a
0000 ;              cluster (routine IOC$CVT_DEVNAM).
0000 ;
0000 ;      V03-012 TCM0002         Trudy C. Matthews       09-Jun-1983
0000 ;              Fix bug in TCM0001.
0000 ;
0000 ;      V03-011 TCM0001         Trudy C. Matthews       21-Apr-1983
0000 ;              Add new parameter to IOC$CVT_DEVNAM that allows caller
0000 ;              to specify whether he wants the node name returned for
0000 ;              local devices or not.
0000 ;
0000 ;      V03-010 ROW0188         Ralph O. Weber          30-APR-1983
0000 ;              Fix broken branches to PMS$ routines.
0000 ;
0000 ;      V03-009 KTA3022         Kerbey T. Altmann       29-Dec-1982
0000 ;              Enhance KTA3018. Add new routine to scan the IO
0000 ;              data base and return the blocks.
0000 ;
0000 ;      V03-008 ROW0140         Ralph O. Weber          18-NOV-1982
0000 ;              Cause IOC$DALOCUBAMAP to give non-fatal INCONSTATE,
0000 ;              "Inconsistant UBA data base" bugcheck if number of map
```

```
0000   ;               registers to deallocate is zero.
0000   ;
0000   ;       V03-007 MLJ0101         Martin L. Jack          11-Nov-1982
0000   ;               Add $SBDEF.
0000   ;
0000   ;       V03-006 KTA3018         Kerbey T. Altmann       01-Nov-1982
0000   ;               Modify CVT_DEVNAME for new IO database.
0000   ;
0000   ;       V03-005 ROW0130         Ralph O. Weber           5-OCT-1982
0000   ;               Remove IOC$DELMBX whose functionality is replaced by new
0000   ;               routines in module UCBCREDEL.
0000   ;
0000   ;       V03-004 KDM0002         Kathleen D. Morse       28-Jun-1982
0000   ;               Added $DCDEF.
0000   ;
0000   ;       V03-003 RLR0003         Robert L. Rappaport     1-June-1982
0000   ;               Correct errors in UNIBUS map register allocation and
0000   ;               deallocation that occur when the number of active
0000   ;               descriptors is zero.  Errors were in IOC$ALOUBAMAPSP
0000   ;               (allocation error), IOC$ALOUBAPRM (allocation error),
0000   ;               and IOC$DALOCUBAMAP (deallocation error).  The error
0000   ;               in IOC$DALOCUBAMAP is corrected in a patch to V3.1.
0000   ;
0000   ;       V03-002 RLR0002         Robert L. Rappaport     22-May-1982
0000   ;               Remove IOC$REQMAPREGN and all comments that reference it.
0000   ;
0000   ;       V03-001 RLR0001 Robert L. Rappaport     22-May-1982
0000   ;               Correct error in UNIBUS map register allocation that
C000   ;               doubly allocated registers when the number of active
0000   ;               descriptors was zero.
0000   ;               This bug corrected in patch to V3.1.
0000   ;
```

```
0000        ;
0000        ;
0000        ; MACRO LIBRARY CALLS
0000        ;
0000
0000                $ADPDEF                        ;DEFINE ADP OFFSETS
0000                $CADEF                         ;DEFINE CONDITIONAL ASSEMBLY PARAMETERS
0000                $CANDEF                        ;DEFINE CANCEL I/O REASON CODES
0000                $CDRPDEF                       ;DEFINE CLASS DRIVER I/O REQUEST PACKET
0000                $CRBDEF                        ;DEFINE CRB OFFSETS
0000                $DCDEF                         ;DEFINE DEVICE CLASSES
0000                $DDBDEF                        ;DEFINE DDB OFFSETS
0000                $DDTDEF                        ;DEFINE DDT OFFSETS
0000                $DEVDEF                        ;DEFINE DEVICE CHARACTERISTICS FLAGS
0000                $DYNDEF                        ;DEFINE DYNAMIC POOL BLOCK TYPES
0000                $EMBDEF                        ;DEFINE EMB OFFSETS
0000                $IDBDEF                        ;DEFINE IDB OFFSETS
0000                $IOCDEF                        ;DEFINE IOC$SEARCHxxx FLAGS
0000                $IPLDEF                        ;DEFINE INTERRUPT PRIORITY LEVELS
0000                $IRPDEF                        ;DEFINE IRP OFFSETS
0000                $JIBDEF                        ;DEFINE JIB OFFSETS
0000                $LCKDEF                        ;DEFINE LOCK MANAGER SYMBOLS
0000                $MSCPDEF                       ;DEFINE MSCP STRUCTURES
0000                $PCBDEF                        ;DEFINE PCB OFFSETS
0000                $PDTDEF                        ;Define PDT offsets
0000                $PRDEF                         ;DEFINE PROCESSOR REGISTERS
0000                $PRVDEF                        ;DEFINE PRIVILEGE BITS
0000                $SBDEF                         ; Define system block offsets
0000                $SSDEF                         ;DEFINE SYSTEM STATUS CODES
0000                $TTYDEF                        ;DEFINE TERMINAL WRITE PACKET OFFSETS
0000                $UBMDDEF                       ;Define UNIBUS Map Descriptor structure
0000                $UCBDEF                        ;DEFINE UCB OFFSETS
0000                $VECDEF                        ;DEFINE CRB VECTOR OFFSETS
```

```
                              0000           .SBTTL  CANCEL I/O ON CHANNEL
                              0000  ;+
                              0000  ; IOC$CANCELIO - CANCEL I/O ON CHANNEL
                              0000  ;
                              0000  ; THIS ROUTINE IS A DEVICE INDEPENDENT CANCEL I/O ROUTINE THAT CONDITIONALLY
                              0000  ; MARKS THE UCB SUCH THAT THE CURRENT I/O REQUEST WILL BE CANCELED IF CONDITIONS
                              0000  ; WARRANT SUCH A ACTION.
                              0000  ;
                              0000  ; INPUTS:
                              0000  ;
                              0000  ;        R2 = NEGATIVE OF THE CHANNEL NUMBER.
                              0000  ;        R3 = CURRENT IO PACKET.
                              0000  ;        R4 = PCB ADDRESS.
                              0000  ;        R5 = UCB ADDRESS.
                              0000  ;
                              0000  ; OUTPUTS:
                              0000  ;
                              0000  ;        IF THE DEVICE IS BUSY, THE REQUEST IS FOR THE CURRENT PROCESS, AND
                              0000  ;        THE I/O WAS ISSUED FROM THE DESIGNATED CHANNEL, THEN THE CANCEL I/O
                              0000  ;        BIT IS SET IN THE CORRESPONDING UCB.
                              0000  ;
                              0000  ;        R2, R3, R4, AND R5 ARE PRESERVED ACROSS CALL.
                              0000  ;-
                              0000
                          00000000           .PSECT  WIONONPAGED
                              0000  IOC$CANCELIO::                                    ;CANCEL I/O ON CHANNEL
11 64 A5     08      E1       0000           BBC     #UCB$V_BSY,UCB$W_STS(R5),10$ ;IF CLR, DEVICE NOT BUSY
60 A4     0C A3      D1       0005           CMPL    IRP$L_PID(R3),PCB$L_PID(R4) ;PROCESS ID MATCH?
             0A      12       000A           BNEQ    10$                          ;IF NEQ NO
   28 A3     52      B1       000C           CMPW    R2,IRP$W_CHAN(R3)            ;CHANNEL NUMBER MATCH
             04      12       0010           BNEQ    10$                          ;IF NEQ NO
   64 A5     08      A8       0012           BISW    #UCB$M_CANCEL,UCB$W_STS(R5) ;SET CANCEL PENDING
                     05       0016  10$:     RSB                                  ;
```

```
                          0017          .SBTTL  Handle Last Channel Deassign
                          0017
                          0017  ;+
                          0017  ; IOC$LAST_CHAN - Last Channel Deassign Specific
                          0017  ; IOC$LAST_CHAN_AMBX - Last Assoc. MBX Channel Deassign Specific
                          0017  ;
                          0017  ; Functional Description:
                          0017  ;
                          0017  ;       Common functions done on last channel deassignment are handled.  The
                          0017  ;       driver's cancel I/O routine is called with an appropriate reason code
                          0017  ;       (CAN$C_DASSGN for regular deassign, or CAN$C_AMBXDGN for associated
                          0017  ;       mailboxes).  If after the cancel routine finished UCB$V_DELETEUCB is
                          0017  ;       set, the UCB is credited and deleted.
                          0017  ;
                          0017  ; Inputs:
                          0017  ;
                          0017  ;       R5      UCB address
                          0017  ;       R2      Channel index (LAST_CHAN only)
                          0017  ;
                          0017  ; Outputs:
                          0017  ;
                          0017  ;       R0 thru R3 destroyed.
                          0017  ;       If appropriate, UCB is deallocated.
                          0017  ;
                          0017  ;-
                          0017
                          0017          .ENABLE LSB
                          0017
                          0017  IOC$LAST_CHAN_AMBX::
              58     DD   0017          PUSHL   R8                      ; Save R8
              52     7C   0019          CLRQ    R2                      ; Clear unused cancel inputs.
         58   02     9A   001B          MOVZBL  #CAN$C_AMBXDGN, R8      ; Set cancel reason code.
              09     11   001E          BRB     10$
                          0020
                          0020  IOC$LAST_CHAN::
              58     DD   0020          PUSHL   R8                      ; Save R8
      53   58 A5     D0   0022          MOVL    UCB$L_IRP(R5), R3       ; Get active packet address.
         58   01     9A   0026          MOVZBL  #CAN$C_DASSGN, R8       ; Set cancel reason code.
                          0029
    50   0088 C5     D0   0029  10$:    MOVL    UCB$L_DDT(R5), R0       ; Get DDT address.
                          002E          SETIPL  UCB$B_FIPL(R5)          ; Raise to fork IPL.
         0C B0       16   0032          JSB     @DDT$L_CANCEL(R0)       ; Call driver's cancel I/O routine.
                          0035          SETIPL  #IPL$_ASTDEL            ; Lower IPL.
   1A 38 A5   17     E0   0038          BBS     #DEV$V_ALL, -           ; Branch if still allocated
                          003D                  UCB$L_DEVCHAR(R5),30$
38 A5  00100004 8F   D3   003D          BITL    #DEV$M_TRM!DEV$M_MBX, - ; Is this a terminal, remote terminal
                          0045                  UCB$L_DEVCHAR(R5)       ; or mailbox?
              05     13   0045          BEQL    20$                     ; Branch if not.
   00 38 A5   07     E4   0047          BBSC    #DEV$V_OPR, -           ; Else, clear OPR bit.
                          004C                  UCB$L_DEVCHAR(R5), 20$  ; This is an implicit operator disable.
              10     E1   004C  20$:    BBC     #UCB$V_DELETEUCB, -     ; Branch if UCB not to be deleted.
        06 64 A5          004E                  UCB$L_STS(R5), 30$
           FFAC'     30   0051          BSBW    IOC$CREDIT_UCB          ; Else credit UCB quotas,
           FFA9'     30   0054          BSBW    IOC$DELETE_UCB          ; and delete the UCB.
           58 8ED0        0057  30$:    POPL    R8                      ; Restore R8
                     05   005A          RSB
                          005B
        005B   330          .DISABLE LSB
```

```
                               005B          .SBTTL  FILL DIAGNOSTIC BUFFER
                               005B   ;+
                               005B   ; IOC$DIAGBUFILL - FILL DIAGNOSTIC BUFFER
                               005B   ;
                               005B   ; THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION, BUT BEFORE RELEASING
                               005B   ; THE I/O CHANNEL, TO FILL THE FINAL DEVICE PARAMETERS INTO AN INTERNAL DIAG-
                               005B   ; NOSTIC BUFFER IF ONE IS SPECIFIED.
                               005B   ;
                               005B   ; INPUTS:
                               005B   ;
                               005B   ;       R4 = ADDRESS OF DEVICE CSR REGISTER.
                               005B   ;       R5 = DEVICE UNIT UCB ADDRESS.
                               005B   ;
                               005B   ; OUTPUTS:
                               005B   ;
                               005B   ;       IF A DIAGNOSTIC BUFFER WAS SPECIFIED IN THE ORIGINAL REQUEST, THEN
                               005B   ;       THE COMPLETION TIME, FINAL ERROR COUNTERS, AND DEVICE REGISTERS ARE
                               005B   ;       FILLED INTO THE DIAGNOSTIC BUFFER.
                               005B   ;-
                               005B
                               005B   IOC$DIAGBUFILL::                                    ;FILL DIAGNOSTIC BUFFER
           53    58 A5    D0   005B          MOVL    UCB$L_IRP(R5),R3              ;GET ADDRESS OF I/O PACKET
       1B 2A A3    07    E1   005F          BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ ;IF CLR, NO DIAGNOSTIC BUFFER
           50    4C B3    D0   0064          MOVL    @IRP$L_DIAGBUF(R3),R0         ;GET ADDRESS OF INTERNAL BUFFER DATA AREA
                 50    08    C0   0068          ADDL    #8,R0                         ;POINT PAST START TIME
   80   00000000'EF    7D   006B          MOVQ    EXE$GQ_SYSTIME,(R0)+          ;INSERT COMPLETION TIME
         80    0080 C5    3C   0072          MOVZWL  UCB$B_ERTCNT(R5),(R0)+        ;INSERT FINAL ERROR COUNTERS
         52    0088 C5    D0   0077          MOVL    UCB$L_DDT(R5),R2              ;GET ADDRESS OF DDT
                 10 B2    16   007C          JSB     @DDT$L_REGDUMP(R2)            ;CALL DEVICE SPECIFIC REGISTER DUMP ROUTINE
                       05   007F   10$:    RSB                                   ;
```

```
                        0080            .SBTTL  RELEASE I/O CHANNEL
                        0080    ;+
                        0080    ; IOC$RELCHAN - RELEASE ALL I/O CHANNELS
                        0080    ; IOC$RELSCHAN - RELEASE SECONDARY I/O CHANNEL
                        0080    ;
                        0080    ; THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION TO RELEASE ALL
                        0080    ; CHANNELS THE I/O WAS BEING PERFORMED ON.
                        0080    ;
                        0080    ; INPUTS:
                        0080    ;
                        0080    ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                        0080    ;
                        0080    ; OUTPUTS:
                        0080    ;
                        0080    ;       THE CHANNELS ARE RELEASED AND AN ATTEMPT IS MADE TO REMOVE THE NEXT
                        0080    ;       WAITING DRIVER PROCESS FROM EACH CHANNEL QUEUE. IF A DRIVER PROCESS
                        0080    ;       IS WAITING, THEN THE CHANNEL IS ASSIGNED TO THAT DRIVER PROCESS AND
                        0080    ;       IT IS CALLED VIA A JSB TO ITS CHANNEL WAIT RETURN ADDRESS. WHEN THE
                        0080    ;       CALLED DRIVER PROCESS RETURNS, A RETURN IS MADE TO THE DRIVER PROCESS
                        0080    ;       THAT RELEASED THE CHANNEL. IF THERE IS NO DRIVER PROCESS WAITING FOR
                        0080    ;       THE CHANNEL, THEN THE CHANNEL STATUS IS SET TO IDLE.
                        0080    ;
                        0080    ;       R3 AND R4 ARE PRESERVED ACROSS CALL.
                        0080    ;-
                        0080
                        0080            .ENABL  LSB
                        0080    IOC$RELSCHAN::                                  ;RELEASE SECONDARY I/O CHANNEL
     50    24 A5   D0   0080            MOVL    UCB$L_CRB(R5),R0                ;GET ADDRESS OF PRIMARY CRB
     50    20 A0   D0   0084            MOVL    CRB$L_LINK(R0),R0               ;GET ADDRESS OF SECONARY CRB
              10   11   0088            BRB     20$                             ;
                        008A    IOC$RELCHAN::                                   ;RELEASE I/O CHANNEL
     50    24 A5   D0   008A            MOVL    UCB$L_CRB(R5),R0                ;GET ADDRESS OF PRIMARY CRB
     50    20 A0   D0   008E            MOVL    CRB$L_LINK(R0),R0               ;GET ADDRESS OF SECONDARY CRB
              02   13   0092            BEQL    10$                             ;IF EQL NONE
              04   10   0094            BSBB    20$                             ;RELEASE SECONDARY CHANNEL
     50    24 A5   D0   0096    10$:    MOVL    UCB$L_CRB(R5),R0                ;GET ADDRESS OF PRIMARY CRB
  25 0E A0     00  E1   009A    20$:    BBC     #CRB$V_BSY,CRB$B_MASK(R0),30$ ;IF CLR, THEN CHANNEL NOT BUSY
     51    2C A0   D0   009F            MOVL    CRB$L_INTD+VEC$L_IDB(R0),R1 ;GET ADDRESS OF IDB
     04 A1     55  D1   00A3            CMPL    R5,IDB$L_OWNER(R1)              ;DRIVER PROCESS OWN CHANNEL?
              1B   12   00A7            BNEQ    30$                             ;IF NEQ NO
     52    00 B0   0F   00A9            REMQUE  @CRB$L_WQFL(R0),R2              ;GET ADDRESS OF NEXT DRIVER FORK BLOCK
              16   1D   00AD            BVS     40$                             ;IF VS NO DRIVER PROCESS WAITING
              38   BB   00AF            PUSHR   #^M<R3,R4,R5>                   ;SAVE CONTEXT OF CURRENT DRIVER PROCESS
        55     52  D0   00B1            MOVL    R2,R5                           ;COPY ADDRESS OF DRIVER PROCESS FORK BLOCK
     53    10 A5   D0   00B4            MOVL    UCB$L_FR3(R5),R3                ;LOAD WAITING DRIVER PROCESS CONTEXT
        54     61  D0   00B8            MOVL    IDB$L_CSR(R1),R4                ;SET ASSIGNED CHANNEL CSR ADDRESS
     04 A1     55  D0   00BB            MOVL    R5,IDB$L_OWNER(R1)              ;SET ADDRESS OF OWNER PROCESS UCB
        0C B5  16       00BF            JSB     @UCB$L_FPC(R5)                  ;CALL DRIVER AT CHANNEL WAIT RETURN ADDRESS
              38   BA   00C2            POPR    #^M<R3,R4,R5>                   ;RESTORE PREVIOUS DRIVER PROCESS CONTEXT
                   05   00C4    30$:    RSB                                     ;
        04 A1      D4   00C5    40$:    CLRL    IDB$L_OWNER(R1)                 ;CLEAR OWNER UNIT UCB ADDRESS
     0E A0     01  8A   00C8            BICB    #CRB$M_BSY,CRB$B_MASK(R0) ;CLEAR CHANNEL BUSY
                   05   00CC            RSB                                     ;
                        00CD            .DSABL  LSB
```

```
                         00CD           .SBTTL  REQUEST I/O CHANNEL
                         00CD      ;+
                         00CD      ; IOC$REQPCHANH - REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
                         00CD      ; IOC$REQSCHANH - REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
                         00CD      ; IOC$REQPCHANL - REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
                         00CD      ; IOC$REQSCHANL - REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
                         00CD      ;
                         00CD      ; THESE ROUTINES ARE CALLED TO REQUEST AN I/O CHANNEL TO PERFORM AN I/O
                         00CD      ; OPERATION ON.
                         00CD      ;
                         00CD      ; INPUTS:
                         00CD      ;
                         00CD      ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                         00CD      ;       04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                         00CD      ;
                         00CD      ; OUTPUTS:
                         00CD      ;
                         00CD      ;       IF THE SPECIFIED I/O CHANNEL IS IDLE, THEN IT IS IMMEDIATELY
                         00CD      ;       ASSIGNED TO THE CURRENT DRIVER PROCESS. ELSE THE DRIVER PROCESS
                         00CD      ;       CONTEXT IS SAVED IN ITS FORK BLOCK, THE FORK BLOCK IS INSERTED
                         00CD      ;       IN THE CHANNEL WAIT QUEUE, AND A RETURN TO THE DRIVER PROCESS'
                         00CD      ;       CALLER IS EXECUTED.
                         00CD      ;
                         00CD      ;       WHEN THE CHANNEL IS ASSIGNED, THE CSR ADDRESS OF THE ASSIGNED
                         00CD      ;       CONTROLLER IS RETURNED TO THE CALLER IN REGISTER R4.
                         00CD      ;
                         00CD      ;       R3 IS PRESERVED ACROSS CALL.
                         00CD      ;-
                         00CD
                         00CD           .ENABL  LSB
                         00CD  IOC$REQSCHANH::                                 ;REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
   50    24 A5     D0    00CD           MOVL    UCB$L_CRB(R5),R0              ;GET ADDRESS OF PRIMARY CRB
   50    20 A0     D0    00D1           MOVL    CRB$L_LINK(R0),R0             ;GET ADDRESS OF SECONDARY CRB
            0E     11    00D5           BRB     10$                           ;
                         00D7  IOC$REQSCHANL::                                 ;REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
   50    24 A5     D0    00D7           MOVL    UCB$L_CRB(R5),R0              ;GET ADDRESS OF PRIMARY CRB
   50    20 A0     D0    00DB           MOVL    CRB$L_LINK(R0),R0             ;GET ADDRESS OF SECONDARY CRB
            0D     11    00DF           BRB     20$                           ;
                         00E1  IOC$REQPCHANH::                                 ;REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
   50    24 A5     D0    00E1           MOVL    UCB$L_CRB(R5),R0              ;GET ADDRESS OF PRIMARY CRB
      52    50     D0    00E5  10$:     MOVL    R0,R2                         ;SET ADDDRESS OF WAIT QUEUE LISTHEAD
            08     11    00E8           BRB     30$                           ;
                         00EA  IOC$REQPCHANL::                                 ;REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
   50    24 A5     D0    00EA           MOVL    UCB$L_CRB(R5),R0              ;GET ADDRESS OF PRIMARY CRB
   52    04 A0     D0    00EE  20$:     MOVL    CRB$L_WQBL(R0),R2             ;GET ADDRESS OF LAST ENTRY IN QUEUE
   51    2C A0     D0    00F2  30$:     MOVL    CRB$L_INTD+VEC$L_IDB(R0),R1 ;GET ADDRESS OF IDB
08 0E A0     00    E2    00F6           BBSS    #CRB$V_BSY,CRB$B_MASK(R0),40$ ;IF SET, THEN CHANNEL BUSY
      54    61     D0    00FB           MOVL    IDB$L_CSR(R1),R4              ;SET ASSIGNED CHANNEL CSR ADDRESS
   04 A1    55     D0    00FE           MOVL    R5,IDB$L_OWNER(R1)            ;SET OWNER UCB ADDRESS
            05           0102           RSB                                   ;
   10 A5    53     D0    0103  40$:     MOVL    R3,UCB$L_FR3(R5)              ;SAVE R3 IN FORK BLOCK
      0C A5      8ED0    0107           POPL    UCB$L_FPC(R5)                 ;SAVE CHANNEL WAIT RETURN ADDRESS
      62    65     0E    010B           INSQUE  UCB$L_FQFL(R5),CRB$L_WQFL(R2) ;INSERT DRIVER PROCESS IN CHANNEL WAI
   04 A1    55     D1    010E           CMPL    R5,IDB$L_OWNER(R1)            ;CURRENT DRIVER PROCESS OWNER?
            03     12    0112           BNEQ    50$                           ;IF NEQ, BRANCH TO RETURN
         FF73      31    0114           BRW     IOC$RELCHAN                   ;IF EQL BRW TO RELEASE CHANNELS
                         0117  50$:
```

```
 05  0117   475          RSB                        ;
     0118   476          .DSABL  LSB
```

```
                          0118                .SBTTL  I/O Request Completion Processing for Class Drivers
                          0118
                          0118       ;+
                          0118       ; IOC$ALTREQCOM - I/O Request Complete Alternate Entry.
                          0118       ;
                          0118       ; This routine is entered when an I/O operation is completed on one
                          0118       ;        one of the devices using the disk or tape class drivers.
                          0118       ;        The packet is inserted in the I/O finish queue for I/O post
                          0118       ;        processing.
                          0118       ;
                          0118       ; INPUTS:
                          0118       ;
                          0118       ;        R0 = First  longword of I/O status
                          0118       ;        R1 = Second longword of I/O status
                          0118       ;        R5 = CDRP address
                          0118       ;
                          0118       ; OUTPUTS:
                          0118       ;
                          0118       ;        The I/O packet is inserted in the I/O Post Processing Queue,
                          0118       ;        a Software interrupt is requested to initiate I/O Post
                          0118       ;        Processing.
                          0118       ;-
                          0118
                          0118       IOC$ALTREQCOM::
       53   A0 A5    9E   0118               MOVAB    CDRP$L_IOQFL(R5),R3     ; R3 => IRP section of CDRP. This is
                          011C                                                ;  for compatibility with rest of QIO
                          011C                                                ;  logic.
       55   1C A3    D0   011C               MOVL     IRP$L_UCB(R3),R5        ; R5 => UCB.
            70 A5    D6   0120               INCL     UCB$L_OPCNT(R5)         ; Increment operations completed
                          0123
            15 50    E9   0123               BLBC     R0,20$                  ; LBC implies I/O error, so goto call
                          0126                                                ;  MOUNT VERIFICATION just in case.
                          0126       10$:
       38 A3    50   7D   0126               MOVQ     R0,IRP$L_MEDIA(R3)      ; Save final I/O status in IRP.
                          012A
                          012A               .IF DF   CA$_MEASURE_IOT
                          012A
     00000000'GF     16   012A               JSB      G^PMS$END_IO            ; Insert end of I/O transaction message
                          0130
                          0130               .ENDC
                          0130
 00000000'FF    63   0E   0130               INSQUE   (R3),@L^IOC$GL_PSBL     ; Insert packet in POST process queue
                          0137               SOFTINT  #IPL$_IOPOST            ; Initiate SOFTWARE INTERRUPT
                     05   013A               RSB
                          013B       20$:
     00000000'GF     16   013B               JSB      G^EXE$MOUNTVER          ; If LBC, call MOUNT VERIFICATION.
             E3      11   0141               BRB      10$                     ; Go back to normal flow.
```

```
                           0143          .SBTTL  I/O REQUEST COMPLETION PROCESSING
                           0143  ;+
                           0143  ; IOC$REQCOM - I/O REQUEST COMPLETE
                           0143  ;
                           0143  ; THIS ROUTINE IS ENTERED WHEN AN I/O OPERATION IS COMPLETED ON A
                           0143  ; DEVICE UNIT. THE FINAL I/O STATUS IS STORED IN THE ASSOCIATED I/O
                           0143  ; PACKET AND THE PACKET IS INSERTED IN THE I/O FINISH QUEUE FOR
                           0143  ; I/O POST PROCESSING. DEVICE UNIT BUSY IS CLEARED AND AN ATTEMPT
                           0143  ; IS MADE TO START ANOTHER I/O REQUEST ON THE DEVICE UNIT.
                           0143  ;
                           0143  ; IF THE I/O REQUEST COMPLETED WITH AN ERROR, AND THE DEVICE IS
                           0143  ; A DISK, THEN BRANCH TO THE MOUNT VERIFICATION CODE, WHICH WILL
                           0143  ; DETERMINE IF THE SITUATION REQUIRES MOUNT VERIFICATION.
                           0143  ;
                           0143  ; IF MOUNT VERIFICATION IS IN PROGRESS, NO FURTHER I/O REQUESTS WILL
                           0143  ; BE INITIATED.  THIS HAS A SIDE EFFECT OF KEEPING THE 'BSY' BIT IN
                           0143  ; WHATEVER STATE IT IS CURRENTLY IN.  FOR CONVENTIONAL DISK DRIVERS,
                           0143  ; THE BSY BIT WILL BE LEFT ON, WHICH WILL BLOCK $QIO FROM INITIATING
                           0143  ; ANY NEW I/O ON THE DEVICE.  FOR THE DISK CLASS DRIVER, THE BUSY
                           0143  ; BIT WILL BE OFF, WHICH WILL ALLOW $QIO TO INITIATE NEW I/O.
                           0143  ;
                           0143  ; INPUTS:
                           0143  ;
                           0143  ;       R0 = FIRST LONGWORD OF I/O STATUS.
                           0143  ;       R1 = SECOND LONGWORD OF I/O STATUS.
                           0143  ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                           0143  ;
                           0143  ; OUTPUTS:
                           0143  ;
                           0143  ;       THE I/O PACKET IS INSERTED IN THE I/O POST PROCESSING QUEUE
                           0143  ;       AND DEVICE UNIT BUSY IS CLEARED. A SOFTWARE INTERRUPT IS
                           0143  ;       REQUESTED TO INITIATE I/O POST PROCESSING.
                           0143  ;-
                           0143
                           0143          .ENABL  LSB
                           0143  IOC$REQCOM::                            ;I/O DONE PROCESSING
   1C 64 A5    02      E5  0143          BBCC    #UCB$V_ERLOGIP,UCB$W_STS(R5),10$ ;IF CLR, ERROR LOG NOT IN PROGRESS
    52   0094 C5       D0  0148          MOVL    UCB$L_EMB(R5),R2        ;GET ADDRESS OF ERROR MESSAGE BUFFER
   1A A2   64 A5       B0  014D          MOVW    UCB$W_STS(R5),EMB$W_DV_STS(R2) ;INSERT FINAL DEVICE STATUS
 10 A2   0080 C5       B0  0152          MOVW    UCB$B_ERTCNT(R5),EMB$B_DV_ERTCNT(R2) ;INSERT FINAL ERROR COUNTERS
     12 A2     50      7D  0158          MOVQ    R0,EMB$Q_DV_IOSB(R2)    ;INSERT FINAL I/O STATUS
               50      DD  015C          PUSHL   R0                      ;SAVE R0
            FE9F'      30  015E          BSBW    ERL$RELEASEMB           ;RELEASE ERROR MESSAGE BUFFER
               50 8ED0     0161          POPL    R0                      ;RESTORE R0
     53     58 A5      D0  0164  10$:    MOVL    UCB$L_IRP(R5),R3        ;GET ADDRESS OF I/O PACKET
            70 A5      D6  0168          INCL    UCB$L_OPCNT(R5)         ;INCREMENT OPERATIONS COMPLETED
            2A 50      E9  016B          BLBC    R0,DISKCHK              ;IF I/O ERROR, CHECK FOR DISK DEVICE
                           016E  ;
                           016E  ; DO NOT SAVE THE I/O STATUS IN THE IRP UNTIL IT HAS BEEN DECIDED THAT
                           016E  ; MOUNT VERIFICATION IS NOT NECESSARY.  THIS IS TO AVOID OVERWRITING THE
                           016E  ; PHYSICAL DISK ADDRESS STORED IN THE IRP AT OFFSET IRP$L_MEDIA.
                           016E  ;
     38 A3     50      7D  016E  20$:    MOVQ    R0,IRP$L_MEDIA(R3)      ;STORE FINAL I/O STATUS
                           0172
                           0172          .IF DF  CA$_MEASURE_IOT
                           0172
      00000000'EF      D5  0172          TSTL    L^PMS$GL_IOPFMPDB       ;DATA COLLECTION ENABLED?
```

```
             36   12  0178                  BNEQ    DO_PMS                  ;BRANCH IF YES
                      017A
                      017A                  .ENDC
                      017A
    00000000'FF   63  0E  017A     PMSEND:  INSQUE  (R3),@L^IOC$GL_PSBL     ;INSERT PACKET IN POST PROCESS QUEUE
                      0181                  SOFTINT #IPL$_IOPOST            ;INITIATE SOFTWARE INTERRUPT
             0E   E0  0184                  BBS     #UCB$V_MNTVERIP,-       ;BRANCH IF MOUNT VERIFICATION IN PROGRESS
       2F 64 A5       0186                          UCB$W_STS(R5),MNTVERPNDCHK ;(NOTE THIS LEAVES 'BSY' AS IS)
    53    4C B5   0F  0189     NXTIRP:  REMQUE  @UCB$L_IOQFL(R5),R3     ;REMOVE I/O PACKET FROM DEVICE UNIT QUEUE
             4C   1C  018D                  BVC     IOC$INITIATE            ;IF VC INITIATE NEXT FUNCTION
  64 A5    0100 8F AA 018F                  BICW    #UCB$M_BSY,UCB$W_STS(R5) ;CLEAR UNIT BUSY
                      0195     RELEASE:                                 ;RELEASE ALL CHANNELS
           FEF2   31  0195                  BRW     IOC$RELCHAN             ;
                      0198     ;
                      0198     ; IF THIS IS A DISK DEVICE, CALL THE MOUNT VERIFICATION ROUTINE
                      0198     ; TO DETERMINE IF MOUNT VERIFICATION IS NECESSARY.  IF NOT, CONTROL
                      0198     ; WILL RETURN, AND THE REQUEST WILL BE COMPLETED IN THE NORMAL MANNER.
                      0198     ;
                      0198     DISKCHK:
             01   91  0198                  CMPB    #DC$_DISK,-             ;IS THIS DEVICE A DISK?
          40 A5       019A                          UCB$B_DEVCLASS(R5)      ;
             D0   12  019C                  BNEQ    20$                     ;BRANCH IF NOT
             13   E5  019E                  BBCC    #UCB$V_MNTVERPND,-      ;CHECK FOR MOUNT VERIFICATION PENDING
       05 64 A5       01A0                          UCB$L_STS(R5),30$       ;IF NOT, JUST ENTER MOUNT VERIFICATION
             0E   E5  01A3                  BBCC    #UCB$V_MNTVERIP,-       ;CLEAR IN-PROGRESS BIT BEFORE CALL
       00 64 A5       01A5                          UCB$L_STS(R5),30$       ; SO IT WILL REALLY START
    00000000'GF   16  01A8     30$:     JSB     G^EXE$MOUNTVER          ;START MOUNT VERIFICATION
             BE   11  01AE                  BRB     20$                     ;COMPLETE I/O REQUEST
                      01B0
                      01B0                  .IF DF  CA$_MEASURE_IOT
                      01B0
    00000000'GF   16  01B0     DO_PMS:  JSB     G^PMS$END_IO            ;INSERT END OF I/O TRANSACTION MESSAGE
             C2   11  01B6                  BRB     PMSEND                  ;REJOIN COMMON CODE
                      01B8
                      01B8                  .ENDC
                      01B8     ;
                      01B8     ; THE MOUNT-VERIFICATION-PENDING BIT IS USED TO INDICATE THAT A DISK SHOULD GO
                      01B8     ; INTO MOUNT VERIFICATION AS SOON AS THE CURRENT I/O IS DONE.  THIS IS INTENDED
                      01B8     ; FOR USE IN A CLUSTER TO STALL I/O WHEN QUORUM IS LOST.
                      01B8     ;
                      01B8     MNTVERPNDCHK:
             13   E5  01B8                  BBCC    #UCB$V_MNTVERPND,-      ;CHECK FOR MOUNT VERIFICATION PENDING
       D8 64 A5       01BA                          UCB$L_STS(R5),RELEASE   ;IF NOT, JUST CLEAN UP
             01   91  01BD                  CMPB    #DC$_DISK,-             ;IS THIS DEVICE A DISK?
          40 A5       01BF                          UCB$B_DEVCLASS(R5)      ;
             D2   12  01C1                  BNEQ    RELEASE                 ;BRANCH IF NOT
             0E   E5  01C3                  BBCC    #UCB$V_MNTVERIP,-       ;CLEAR IN-PROGRESS BIT BEFORE CALL
       00 64 A5       01C5                          UCB$L_STS(R5),40$
             53   D4  01C8     40$:     CLRL    R3                      ;NO IRP PASSED TO MOUNT VERIFICATION
    00000000'GF   16  01CA                  JSB     G^EXE$MOUNTVER          ;TRY TO START MOUNT VERIFICATION
             B7   11  01D0                  BRB     NXTIRP                  ;WASN'T NECESSARY
                      01D2
                      01D2                  .DSABL  LSB
```

```
                 01D2          .SBTTL  MOUNT VERIFICATION HELPER
                 01D2   ;++
                 01D2   ; IOC$MNTVER    - Assist driver with mount verification.
                 01D2   ;
                 01D2   ; This routine is called by EXE$MOUNTVER to perform some driver-specific
                 01D2   ; actions necessary for mount verification.  This routine is used by non-
                 01D2   ; CLASS drivers, and is called by default if EXE$MOUNTVER finds the address
                 01D2   ; of IOC$RETURN in DDT$L_MNTVER.
                 01D2   ;
                 01D2   ; Inputs:
                 01D2   ;
                 01D2   ;       R3      = IRP address or 0
                 01D2   ;       R5      = UCB address
                 01D2   ;
                 01D2   ; Outputs:
                 01D2   ;
                 01D2   ;       None.
                 01D2   ;
                 01D2   ; Side effects:
                 01D2   ;
                 01D2   ;       If R3 contains an IRP address, the IRP will be queued to the
                 01D2   ;       head of the UCB's IRP work queue.  If R3 contains is zero, then
                 01D2   ;       remove the IRP from the head of the UCB's work queue and attempt
                 01D2   ;       to initiate the I/O.
                 01D2   ;--
                 01D2
                 01D2   IOC$MNTVER::                            ;Driver-specific mount verification code
      53    D5   01D2           TSTL    R3                      ;Check IRP address
      B3    13   01D4           BEQL    NXTIRP                  ;Branch if none
      63    0E   01D6           INSQUE  IRP$L_IOQFL(R3),-       ;Requeue the IRP
   4C A5         01D8                   UCB$L_IOQFL(R5)         ;
            05   01DA           RSB                             ;Return
```

```
                          01DB             .SBTTL  INITIATE I/O FUNCTION ON DEVICE
                          01DB      ;+
                          01DB      ; IOC$INITIATE - INITIATE NEXT FUNCTION ON DEVICE
                          01DB      ;
                          01DB      ; THIS ROUTINE IS CALLED TO INITIATE THE NEXT FUNCTION ON A DEVICE BY CLEARING
                          01DB      ; STATUS BITS, SETTING THE OPERATION START TIME IF A DIAGNOSTIC BUFFER IS
                          01DB      ; SPECIFIED, AND CALLING THE DRIVER AT ITS START I/O ENTRY POINT.
                          01DB      ;
                          01DB      ; INPUTS:
                          01DB      ;
                          01DB      ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                          01DB      ;       R5 = DEVICE UNIT UCB ADDRESS.
                          01DB      ;
                          01DB      ; OUTPUTS:
                          01DB      ;
                          01DB      ;       CANCEL I/O, POWERFAIL, AND TIME OUT STATUS BITS ARE CLEARED, THE
                          01DB      ;       CURRENT SYSTEM TIME IS FILLED INTO THE INTERNAL DIAGNOSTIC BUFFER
                          01DB      ;       IF ONE IS SPECIFIED, AND THE DRIVER IS CALLED AT ITS START I/O ENTRY
                          01DB      ;       POINT.
                          01DB      ;-
                          01DB
                          01DB      IOC$INITIATE::                              ;INITIATE I/O FUNCTION
       58 A5   53    D0   01DB             MOVL    R3,UCB$L_IRP(R5)            ;SAVE I/O PACKET ADDRESS
                          01DF
                          01DF             .IF DF  CA$_MEASURE_IOT
                          01DF
    00000000'GF      16   01DF             JSB     G^PMS$START_IO              ;INSERT START OF I/O TRANSACTION MESSAGE
                          01E5
                          01E5             .ENDC
                          01E5
    78 A5   2C A3    7D   01E5             MOVQ    IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) ;COPY TRANSFER PARAMETERS
 64 A5   0048 8F     AA   01EA             BICW    #UCB$M_CANCEL!UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR CANCEL AND TIME OUT
  0B 2A A3    07     E1   01F0             BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ ;IF CLR, NO DIAGNOSTIC BUFFER
       50   4C B3    D0   01F5             MOVL    @IRP$L_DIAGBUF(R3),R0       ;GET ADDRESS OF DIAGNOSTIC BUFFER DATA AREA
60   00000000'EF     7D   01F9             MOVQ    EXE$GQ_SYSTIME,(R0)         ;INSERT I/O OPERATION START TIME
     50   0088 C5    D0   0200      10$:   MOVL    UCB$L_DDT(R5),R0            ;GET ADDRESS OF DRIVER DISPATCH TABLE
            00 B0    17   0205             JMP     @DDT$L_START(R0)            ;START I/O OPERATION
```

```
                     0208          .SBTTL  Allocate Buffered Data Path
                     0208  ;+
                     0208  ; ALLOCATE BUFFERED DATA PATH CODE -
                     0208  ;
                     0208  ; IOC$REQDATAP - Entrypoint (called from traditional drivers) where caller
                     0208  ;       wishes to be queued (using UCB fork block) if no buffered data path
                     0208  ;       is available at the time of the call.
                     0208  ; INPUT:
                     0208  ;               R5 => UCB.
                     0208  ;
                     0208  ; IOC$REQDATAPNW - Entrypoint to call when caller does not want to wait for
                     0208  ;       unavailable data path.
                     0208  ; INPUT:
                     0208  ;               R5 => UCB
                     0208  ;
                     0208  ; IOC$REQDATAPUDA - Entrypoint (called from UDA port driver) where CDRP
                     0208  ;       is used as the source of information about the request and where
                     0208  ;       the caller does not want to wait for unavailable datapath.
                     0208  ;
                     0208  ; INPUT:
                     0208  ;               R4 => PDT
                     0208  ;               R5 => CDRP
                     0208  ;-
                     0208
                     0208  IOC$REQDATAP::
            10    10 0208          BSBB    IOC$REQDATAPNW          ; Try to alloc. and get control after.
         0C 50    E8 020A          BLBS    R0,10$                  ; LBS implies allocation success.
                     020D
   10 A5    53    7D 020D          MOVQ    R3,UCB$L_FR3(R5)        ; Save driver context in UCB fork block.
         0C A5  8ED0 0211          POPL    UCB$L_FPC(R5)           ; Save caller's return point.
            65    0E 0215          INSQUE  UCB$L_FQFL(R5),-        ; Queue fork block to resource wait queue.
         18 B1       0217                  @ADP$L_DPQBL(R1)        ;  Assumes IOC$ALODATAP saves R1=>ADP.
                  05 0219  10$:    RSB                             ; Return to caller or caller's caller.
                     021A
                     021A  IOC$REQDATAPNW::
   50    24 A5    D0 021A          MOVL    UCB$L_CRB(R5),R0                ; R0=>CRB.
   51    38 A0    D0 021E          MOVL    CRB$L_INTD+VEC$L_ADP(R0),R1     ; R1=>ADP (pass to IOC$ALODATAP)
   52    34 A0    9E 0222          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R2  ; R2=>UBMD    "    "     "
                     0226
            40    11 0226          BRB     IOC$ALODATAP            ; NOWAIT, RSB from IOC$ALODATAP
                     0228                                          ;  returns to our caller.
                     0228  IOC$REQDATAPUDA::
 51    00E0 C4    D0 0228          MOVL    PDT$L_ADP(R4),R1        ; R1=>ADP (pass to IOC$ALODATAP)
   52    3C A5    9E 022D          MOVAB   CDRP$L_UBARSRCE(R5),R2  ; R2=>UBMD    "    "     "
                     0231
            35    10 0231          BSBB    IOC$ALODATAP            ; Call to allocate a data path.
         2D 50    E8 0233          BLBS    R0,20$                  ; LBS means we got one.
      29 D0 A5    E9 0236          BLBC    CDRP$W_BOFF(R5),20$     ; LBC means, user buffer is on an
                     023A                                          ;  even byte address so we can use
                     023A                                          ;  the Direct Data Path.
                     023A
                     023A  ; Here we have a transfer to a user buffer located at an odd byte address.
                     023A  ; On those processors which support Byte Offset on the Direct Datapath, we
                     023A  ; can continue processing.  On other processors, we must wait for a buffered
                     023A  ; datapath.
                     023A
                     023A          CPUDISP <<780,10$>,-            ; On 11-780 we wait.
                     023A                             <750,20$>,-             ; On 11-750 we continue.
                     023A                             <730,20$>,-             ; On 11-730 we continue.
                     023A                             <790,10$>,-             ; On 11-790 we wait.
                     023A                             <8SS,10$>,-             ; On SCORPIO we wait.
                     023A                             <8NN,10$>,-             ; On NAUTILUS we wait.
                     023A                             <UV1,30$>>              ; On MicroVAX we bugcheck.
                     0254
10 A5    53     7D   0254  10$:    MOVQ     R3,CDRP$L_FR3(R5)       ; Save driver context in CDRP fork block.
   0C    A5   8ED0   0258          POPL     CDRP$L_FPC(R5)          ; Save caller's return point.
   28    B5     B6   025C          INCW     @CDRP$L_RWCPTR(R5)      ; Increment RWAITCNT.
         65     0E   025F          INSQUE   CDRP$L_FQFL(R5),-       ; Queue fork block to resource wait queue.
   18    B1          0261                   @ADP$L_DPQBL(R1)        ;  Assumes IOC$ALODATAP saves R1=>ADP.
                     0263  20$:
                05   0263          RSB                              ; Return to caller or caller's caller.
                     0264
                     0264  30$:    BUG_CHECK        IVBYTEALGN,FATAL
```

```
                         0268 ; IOC$ALODATAP - Common subroutine called by above routines to allocate
                         0268 ;         a UNIBUS buffered datapath.
                         0268 ;
                         0268 ; INPUTS:
                         0268 ;         R1 => ADP wherein the datapath allocation bit map is stored.
                         0268 ;         R2 => UBA mapping descriptor in user's data structure.
                         0268 ;
                         0268 ; OUTPUTS:
                         0268 ;         R0 LBS - implies allocation success
                         0268 ;                 datapath field in R2 => UBA mapping descriptor is set to the
                         0268 ;                         number of the datapath allocated.
                         0268 ;                 appropriate bit in datapath allocation bit map is cleared.
                         0268 ;         R0 LBC - implies allocation failure.
                         0268 ;
                         0268
                         0268 IOC$ALODATAP:
           07      E0    0268           BBS     #VEC$V_PATHLOCK,-         ; If this user has a permanently allocated
     17 03 A2            026A                   UBMD$B_DATAPATH(R2),10$ ;  datapath, branch around to success.
                         026D
                         026D           ASSUME  ADP$C_NUMDATAP  EQ  16
           00      EA    026D           FFS     #0,-                      ; Find first available datapath,
           10            026F                   #ADP$C_NUMDATAP,-         ;  according to bit map.  Note failure
        60 A1            0270                   ADP$W_DPBITMAP(R1),-      ;  leaves R0 with the value "16", an
           50            0272                   R0                        ;  even number with the low bit clear.
           12      13    0273           BEQL    20$                       ; EQL implies failure.
                         0275
           50      F0    0275           INSV    R0,-                      ; Upon success, R0 has number of the
                         0277                   #VEC$V_DATAPATH,-         ;  available datapath to allocate.
     05    00            0277                   #VEC$S_DATAPATH,-         ;  So we update the user's datapath
        03 A2            0279                   UBMD$B_DATAPATH(R2)       ;  descriptor pointed at by R2.
                         027B
 04 60 A1  50      E4    027B           BBSC    R0,ADP$W_DPBITMAP(R1),10$; And we update the bit map.
                         0280           BUG_CHECK INCONSTATE              ; We shouldn't be here obviously.
                         0284
     50    01      D0    0284 10$:      MOVL    S^#SS$_NORMAL,R0          ; Indicate allocation success.
                   05    0287 20$:      RSB                               ; And we return to our caller.
```

```
                           0288         .SBTTL  Release Buffered Data Path
                           0288  ;+
                           0288  ; RELEASE BUFFERED DATA PATH CODE -
                           0288  ;
                           0288  ; IOC$RELDATAPUDA - Entry point called from UDA port driver in response
                           0288  ;         to an UNMAP call.  Here the data as to the buffered data path
                           0288  ;         is in the CDRP.
                           0288  ;
                           0288  ; INPUTS:
                           0288  ;         R4 => PDT
                           0288  ;         R5 => CDRP
                           0288  ;
                           0288  ; IOC$RELDATAP - Entry point called from traditional drivers to release
                           0288  ;         the buffered datapath described in CRB$L_INTD+VEC$B_DATAPATH.
                           0288  ;
                           0288  ; INPUTS:
                           0288  ;         R5 => UCB
                           0288  ;
                           0288  ; OUTPUTS:
                           0288  ;         Datapath re-allocated (if any waiters). R0, R1, and R2 modified.
                           0288  ; NOTE: Since calls to IOC$REQDATAPUDA are NOWAIT, fork blocks dequeued
                           0288  ;         here from ADP$L_DPQFL are guaranteed to be UCB's.
                           0288  ;
                           0288
                           0288 IOC$RELDATAPUDA::
51   00E0 C4      D0       0288         MOVL    PDT$L_ADP(R4),R1        ; R1 => ADP.
  52   3C A5      9E       028D         MOVAB   CDRP$L_UBARSRCE(R5),R2  ; R2 => UBMD.
          0C      11       0291         BRB     RELDATAP_COMMON
                           0293 IOC$RELDATAP::
  50   24 A5      D0       0293         MOVL    UCB$L_CRB(R5),R0                 ; R0 => CRB.
  52   34 A0      9E       0297         MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R2   ; R2 => UBMD.
  51   38 A0      D0       029B         MOVL    CRB$L_INTD+VEC$L_ADP(R0),R1      ; R1 => ADP.
                           029F RELDATAP_COMMON:
  50   03 A2      98       029F         CVTBL   UBMD$B_DATAPATH(R2),R0           ; Get datapath designator.
          36      15       02A3         BLEQ    10$                              ; If LSS permanent assignment.
                           02A5                                                  ; If EQL we had NO datapath to
                           02A5                                                  ;  release.
          00      F0       02A5         INSV    #0,-                             ; Zero datapath number.
      05  00               02A7                 #VEC$V_DATAPATH,#VEC$S_DATAPATH,-
       03 A2               02A9                 UBMD$B_DATAPATH(R2)
          00      EF       02AB         EXTZV   #VEC$V_DATAPATH,-       ; Extract datapath number.
52    50  05               02AD                 #VEC$S_DATAPATH,R0,R2   ;
  50   14 B1      0F       02B0         REMQUE  @ADP$L_DPQFL(R1),R0     ; R0 => next driver fork block
          26      1D       02B4         BVS     20$                     ; If VS no driver process waiting
                           02B6
      7E  53      7D       02B6         MOVQ    R3,-(SP)                ; Save R3, R4, R5
          55      DD       02B9         PUSHL   R5
      55  50      D0       02BB         MOVL    R0,R5                   ; R5 => driver fork block.
          10      91       02BE         CMPB    #DYN$C_UCB,-            ; See if we dequeued a UCB or a CDRP.
       0A A5               02C0                 UCB$B_TYPE(R5)
          22      12       02C2         BNEQ    30$                     ; NEQ implies a CDRP.
                           02C4
                           02C4 ; Here we have R5 => UCB.
                           02C4
  51   24 A5      D0       02C4         MOVL    UCB$L_CRB(R5),R1        ; R1 => CRB.
                           02C8
          52      F0       02C8         INSV    R2,-                    ; Store assigned datapath #
```

```
                         02CA   876                         #VEC$V_DATAPATH,-           ;  in CRB.
          05  00             02CA   877                         #VEC$S_DATAPATH,-
             37 A1           02CC   878                         CRB$L_INTD+VEC$B_DATAPATH(R1)
                         02CE   879
     53   10 A5   7D    02CE   880             MOVQ     UCB$L_FR3(R5),R3            ; Restore driver context.
          0C B5   16    02D2   881             JSB      @UCB$L_FPC(R5)              ; Call back waiting driver.
                         02D5   882 5$:
             55 8ED0    02D5   883             POPL     R5                          ; Restore deallocator's R5,R4,R3
     53      8E   7D    02D8   884             MOVQ     (SP)+,R3                    ;
                  05    02DB   885 10$:        RSB                                  ; Return to deallocator.
 FA 60 A1    52   E3    02DC   886 20$:        BBCS     R2,-
                         02E1   887                      ADP$W_DPBITMAP(R1),10$      ; Set datapath bit and exit
                         02E1   888             BUG_CHECK INCONSTATE                 ; Inconsistent state.
                  05    02E5   889             RSB                                  ;
                         02E6   890
                         02E6   891 ; Here we have R5 => CDRP.
                         02E6   892
                         02E6   893 30$:
             52   F0    02E6   894             INSV     R2,-                        ; Store assigned datapath #
                         02E8   895                      #VEC$V_DATAPATH,-           ;  in CDRP field.
          05  00             02E8   896                      #VEC$S_DATAPATH,-
             3F A5           02EA   897                      CDRP$L_UBARSRCE+UBMD$B_DATAPATH(R5)
                         02EC   898
   00000000'EF    16    02EC   899             JSB      SCS$RESUMEWAITR             ; Resume waiting thread and any backed
                         02F2   900                                                  ;  up IRP's.
             E1   11    02F2   901             BRB      5$                          ; Branch back to resume deallocator's
                         02F4   902                                                  ;  thread.
```

```
                           02F4              .SBTTL  REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
                           02F4      ;+
                           02F4      ; IOC$REQMAPUDA - REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
                           02F4      ;
                           02F4      ; THIS ROUTINE IS CALLED TO ALLOCATE UBA MAP REGISTERS AND TO MARK THE ALLOCATION
                           02F4      ; IN THE UBA MAP REGISTER ALLOCATION DATA STRUCTURES.
                           02F4      ;
                           02F4      ; INPUTS:
                           02F4      ;
                           02F4      ;       R4 = ADDRESS OF PORT DESCRIPTOR TABLE.
                           02F4      ;       R5 = ADDRESS OF CLASS DRIVER REQUEST PACKET (CDRP).
                           02F4      ;
                           02F4      ; OUTPUTS:
                           02F4      ;
                           02F4      ;       IF MAP REGISTERS ARE ALLOCATED FOR THE CDRP, THE APPROPRIATE FIELDS
                           02F4      ;       IN THE CDRP ARE MODIFIED TO INDICATE WHICH REGISTERS, AND THE NUMBER
                           02F4      ;       OF REGISTERS THAT HAVE BEEN ALLOCATED.  ALSO THE ALLOCATION DATA
                           02F4      ;       STRUCTURE IN THE ADP IS MODIFIED.
                           02F4      ;
                           02F4      ;       IF MAP REGISTERS CANNOT BE ALLOCATED AT THIS TIME, THE CDRP IS
                           02F4      ;       QUEUED ONTO THE RESOURCE WAIT LIST AND THE UCB$W_RWAITCNT IS
                           02F4      ;       INCREMENTED.
                           02F4      ;
                           02F4      ;-
                           02F4
                           02F4      IOC$REQMAPUDA::                                  ; Allocate UBA map registers for class drive
              25    10     02F4              BSBB    IOC$ALOMAPUDA                    ; Call to allocate map registers if availabl
                           02F6                                                       ;  Returns R2 => ADP.
                           02F6
                           02F6      ; If here, low bit of R0 tells us whether we were successful in the allocation
                           02F6      ;       attempt.
                           02F6
           0F 50    E8     02F6              BLBS    R0,10$                           ; Branch around if successful.
        10 A5  53   7D     02F9              MOVQ    R3,CDRP$L_FR3(R5)                ; Save driver process context
           28 B5    B6     02FD              INCW    @CDRP$L_RQCPTR(R5)               ; One more CDRP, on this UCB, awaiting
                           0300                                                       ;  resources.
           0C A5  8ED0     0300              POPL    CDRP$L_FPC(R5)                   ; Save map register wait return address
              65    0E     0304              INSQUE  CDRP$L_FQFL(R5),-
           34 B2           0306                      @ADP$L_MRQBL(R2)                 ; Insert process in map register wait queue
                    05     0308      10$:    RSB                                      ;
```

```
                         0309         .SBTTL  REQUEST UNIBUS MAP REGISTERS
                         0309 ;+
                         0309 ; IOC$REQMAPREG - REQUEST UNIBUS MAP REGISTERS
                         0309 ;
                         0309 ; THIS ROUTINE IS CALLED TO REQUEST UNIBUS MAP REGISTERS TO PERFORM AN
                         0309 ; I/O TRANSFER.
                         0309 ;
                         0309 ; INPUTS:
                         0309 ;
                         0309 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                         0309 ;       04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                         0309 ;
                         0309 ;       IT IS ASSUMED THAT THE CALLER OWNS THE I/O CHANNEL ON WHICH THE
                         0309 ;       TRANSFER IS TO OCCUR ON.
                         0309 ;
                         0309 ; OUTPUTS:
                         0309 ;
                         0309 ;       IF MAP REGISTERS HAVE BEEN PERMANENTLY ASSIGNED TO THE ASSOCIATED
                         0309 ;       I/O CHANNEL, THEN CONTROL IS IMMEDIATELY RETURNED TO THE CALLER.
                         0309 ;       ELSE AN ATTEMPT IS MADE TO ALLOCATE THE REQUESTED NUMBER OF MAP REG-
                         0309 ;       ISTERS. IF SUFFICIENT CONTIGUOUS MAP REGISTERS ARE FOUND, THEN THEY
                         0309 ;       ARE ASSIGNED TO THE ASSOCIATED I/O CHANNEL AND CONTROL IS RETURNED
                         0309 ;       TO THE CALLER. ELSE THE DRIVER PROCESS CONTEXT IS SAVED IN ITS FORK
                         0309 ;       BLOCK, THE FORK BLOCK IS INSERTED IN THE MAP REGISTER WAIT QUEUE,
                         0309 ;       AND A RETURN TO THE DRIVER PROCESS' CALLER IS EXECUTED.
                         0309 ;-
                         0309
                         0309 IOC$REQMAPREG::                              ;REQUEST UNIBUS MAP REGISTERS
            3A    10     0309         BSBB    IOC$ALOUBAMAP                ; ALLOCATE UBA MAP REGISTER
      0C    50    E8     030B         BLBS    R0,10$                       ;IF LBS SUCCESSFUL ALLOCATION
10 A5       53    7D     030E         MOVQ    R3,UCB$L_FR3(R5)             ;SAVE DRIVER PROCESS CONTEXT
      0C    A5  8ED0     0312         POPL    UCB$L_FPC(R5)                ;SAVE MAP REGISTER WAIT RETURN ADDRESS
34 B2       65    0E     0316         INSQUE  UCB$L_FQFL(R5),@ADP$L_MRQBL(R2) ;INSERT PROCESS IN MAP REGISTER WAIT
                  05     031A 10$:    RSB                                  ;
```

```
                          031B         .SBTTL  ALLOCATE UNIBUS MAP REGISTERS
                          031B  ;+
                          031B  ; IOC$ALOUBAMAP - ALLOCATE UBA MAP REGISTERS (CRB DATABASE SPECIFIED)
                          031B  ; IOC$ALOUBAMAPN - ALLOCATE UBA MAP REGISTERS (ARGUMENT SPECIFIED)
                          031B  ; IOC$ALOMAPUDA - ALLOCATE UBA MAP REGISTERS (FOR CLASS DRIVER(S))
                          031B  ;
                          031B  ; This routine is called to allocate uba map registers and to mark the allocation
                          031B  ;       in the map register allocation structure located in the ADP.  The state
                          031B  ;       of the UNIBUS map registers is maintained in a set of descriptors
                          031B  ;       that describe contiguous extents of allocatable (i.e. free) map
                          031B  ;       registers.  A map register descriptor consists of the
                          031B  ;       corresponding elements of two distinct arrays (of one word items)
                          031B  ;       located in the ADP.  These arrays, ADP$W_MRNREGARY and ADP$W_MRFREGARY,
                          031B  ;       contain the number of map registers and the first map register in each
                          031B  ;       contiguous extent of free map registers.  These arrays are each
                          031B  ;       preceeded by a one word field containing all 1's (-1) so that compares
                          031B  ;       made against the "previous" descriptor fail when the current desciptor
                          031B  ;       is the one whose index is zero.
                          031B  ;
                          031B  ;       ADP$L_MRACTMDRS maintains the number of active descriptors, i.e. the
                          031B  ;       number of elements of each array which contain valid data.
                          031B  ;
                          031B  ; INPUTS: (FOR IOC$ALOUBAMAP AND ALOUBAMAPN)
                          031B  ;       R3 = NUMBER OF MAP REGISTERS TO ALLOCATE (IOC$ALOUBAMAPN only).
                          031B  ;       R5 = DEVICE UNIT UCB ADDRESS.
                          031B  ;
                          031B  ; INPUT:  (FOR IOC$ALOMAPUDA)
                          031B  ;       R4 => PDT
                          031B  ;       R5 => CDRP
                          031B  ;
                          031B  ; OUTPUTS:
                          031B  ;       R0 = SUCCESS INDICATION.
                          031B  ;       R2 => ADP
                          031B  ;-
                          031B          .enabl  lsb
                          031B  IOC$ALOMAPUDA:
        7E    53    7D    031B          MOVQ    R3,-(SP)                ; Save R3,R4,R5
              55    DD    031E          PUSHL   R5                      ;
                          0320
    52   00E0 C4    D0    0320          MOVL    PDT$L_ADP(R4),R2        ; R2 => ADP before we modify R4.
                          0325
     53    D2 A5    D0    0325          MOVL    CDRP$L_BCNT(R5),R3      ; Get transfer byte count
     54    D0 A5    3C    0329          MOVZWL  CDRP$W_BOFF(R5),R4      ; Get byte offset in page
  53   03FF C344    9E    032D          MOVAB   ^X3FF(R3)[R4],R3        ; Calculate highest relative byte and round
 53   53    F7 8F   78    0333          ASHL    #-9,R3,R3               ; Calculate number of map registers required
                          0338
     51    3C A5    9E    0338          MOVAB   CDRP$L_UBARSRCE(R5),R1  ; R1 => UBMD.
              2F    11    033C          BRB     COMMON_ALOUBAMAP        ; Branch to common code.
                          033E
                          033E  IOC$ALOUBAMAPN::                        ;ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFI
        7E    53    7D    033E          MOVQ    R3,-(SP)                ; Save R3,R4,R5
              55    DD    0341          PUSHL   R5                      ;
              18    11    0343          BRB     5$                      ;
                          0345
                          0345  IOC$ALOUBAMAP::                         ;ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
        7E    53    7D    0345          MOVQ    R3,-(SP)                ; Save R3,R4,R5
              55    DD    0348          PUSHL   R5                      ;
```

```
                                  034A
          53   7E A5     3C  034A              MOVZWL  UCB$W_BCNT(R5),R3          ;GET TRANSFER BYTE COUNT
          54   7C A5     3C  034E              MOVZWL  UCB$W_BOFF(R5),R4          ;GET BYTE OFFSET IN PAGE
     53   03FF C344     9E  0352              MOVAB   ^X3FF[R3][R4],R3           ;CALCULATE HIGHEST RELATIVE BYTE AND ROUND
  53      53   F7 8F     78  0358              ASHL    #-9,R3,R3                  ;CALCULATE NUMBER OF MAP REGISTERS REQUIRED
                             035D        5$:
          51   24 A5     D0  035D              MOVL    UCB$L_CRB(R5),R1                  ; R1 => CRB.
          52   38 A1     D0  0361              MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2       ; R2 => ADP.
          51   34 A1     9E  0365              MOVAB   CRB$L_INTD+VEC$W_MAPREG(R1),R1    ; R1 => UBMD.
                  0F     E0  0369              BBS     #VEC$V_MAPLOCK,-                  ; If SET, already permanently
               38 61         036B                      UBMD$W_MAPREG(R1),40$             ;  allocated, so branch around.
                             036D
                             036D        ; Here:
                             036D        ;         R1 => UBMD - caller's structure where we record registers allocated
                             036D        ;         R2 => ADP
                             036D        ;         R3 =  number of map registers to allocate
                             036D        ;
                             036D
                             036D        COMMON_ALOUBAMAP:
               5C A2     D5  036D              TSTL    ADP$L_MRACTMDRS(R2)        ; Test for zero active descriptors.
                  13     13  0370              BEQL    15$                        ; EQL implies no registers available.
                  53     D6  0372              INCL    R3                         ; Round up request to next multiple
            53    01     8A  0374              BICB    #1,R3                      ;  of 2.
                  55     D4  0377              CLRL    R5                         ; Establish loop variable.
                             0379        10$:
      64 A245     53     B1  0379              CMPW    R3,ADP$W_MRNREGARY(R2)[R5]         ; See if enough regs described here.
                  09     15  037E              BLEQ    20$                                ; LEQ implies YES.
                             0380
   F4 55    5C A2     F2  0380              AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$         ; Else branch back and continue
                             0385        15$:
                  50     D4  0385              CLRL    R0                                 ; If here, allocation failure.
                  1F     11  0387              BRB     50$                                ; Branch around to return.
                             0389        20$:
  61    015E C245     B0  0389              MOVW    ADP$W_MRFREGARY(R2)[R5],-          ; Allocate from low end of extent
                             038F                      UBMD$W_MAPREG(R1)                  ;  by copying 1st map reg. #.
        02 A1     53     90  038F              MOVB    R3,UBMD$B_NUMREG(R1)               ; Set # of map regs allocated.
      64 A245     53     A2  0393              SUBW    R3,ADP$W_MRNREGARY(R2)[R5]         ; Subtract out # regs allocated.
                  05     12  0398              BNEQ    30$                                ; NEQ implies extent not empty,
                             039A                                                         ;   branch around deallocate.
                0129     30  039A              BSBW    DEALLOC_DESCRIP                    ; Call to deallocate descriptor.
                  06     11  039D              BRB     40$                                ;  And branch back to return.
                             039F        30$:
  015E C245     53     A0  039F              ADDW    R3,ADP$W_MRFREGARY(R2)[R5]         ; Bump descriptor past
                             03A5                                                         ;  allocated registers.
            50    01     D0  03A5        40$:  MOVL    S^#SS$_NORMAL,R0           ; Indicate success.
                  55   8ED0  03A8        50$:  POPL    R5                         ; Restore R5,R4,R3
            53    8E     7D  03AB              MOVQ    (SP)+,R3                   ;
                         05  03AE              RSB
                             03AF              .dsabl  lsb
```

```
                                03AF          .SBTTL  Allocate a specific set of UNIBUS Map Registers
                                03AF  ;+
                                03AF  ; IOC$ALOUBAMAPSP
                                03AF  ;
                                03AF  ; This routine is called to allocate a specific set of UNIBUS Map Registers.
                                03AF  ;
                                03AF  ; INPUTS:
                                03AF  ;       R3 =  # of map registers to allocate
                                03AF  ;       R4 =  # of first map register to allocate
                                03AF  ;       R5 => UCB
                                03AF  ;
                                03AF  ; OUTPUTS:
                                03AF  ;       R0 =  Success or failure indication
                                03AF  ;               Note R0, R1 and R2 modified.
                                03AF  ;-
                                03AF
                                03AF  IOC$ALOUBAMAPSP::
                    7E   53   7D  03AF          MOVQ    R3,-(SP)                        ; Save R3,R4,R5
                         55   DD  03B2          PUSHL   R5                              ;
                                03B4
                 50   24 A5   D0  03B4          MOVL    UCB$L_CRB(R5),R0                  ; R0 => CRB.
                 52   38 A0   D0  03B8          MOVL    CRB$L_INTD+VEC$L_ADP(R0),R2       ; R2 => ADP.
                 51   34 A0   9E  03BC          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R1    ; R1 => UBA mapping descriptor.
                                03C0
                      5C A2   D5  03C0          TSTL    ADP$L_MRACTMDRS(R2)     ; Test for zero active descriptors.
                         2C   13  03C3          BEQL    30$                     ; EQL implies no registers available.
                      05 54   E9  03C5          BLBC    R4,10$                  ; Prepare to round DOWN to even boundary.
                    54   01   8A  03C8          BICB    #1,R4                   ; Clear low bit if set and
                         53   D6  03CB          INCL    R3                      ;  then increment # of registers to allocate
                                03CD  10$:
                         53   D6  03CD          INCL    R3                      ; Prepare to round UP to even # of registers
                    53   01   8A  03CF          BICB    #1,R3                   ;
                                03D2
                         55   D4  03D2          CLRL    R5                      ; R5 will be index register.
                                03D4  20$:
             015E C245   54   B1  03D4          CMPW    R4,ADP$W_MRFREGARY(R2)[R5]      ; Are registers we want in
                                03DA                                                  ;  current extent?
                         15   19  03DA          BLSS    30$                     ; LSS means current extent is beyond the
                                03DC                                            ;  desired registers. Therefore they are
                                03DC                                            ;  not availiable and we have failed.
                         46   13  03DC          BEQL    50$                     ; EQL means they are at the beginning
                                03DE                                            ;  of the current extent.
                                03DE
                                03DE  ; Here the registers we want are either within the middle of the current
                                03DE  ;       extent or else beyond the current extent.
                                03DE
   50   64 A245   015E C245   A1  03DE          ADDW3   ADP$W_MRFREGARY(R2)[R5],-         ; R0 = 1st register beyond
                                03E7                  ADP$W_MRNREGARY(R2)[R5],R0        ;  current extent.
                    50   54   B1  03E7          CMPW    R4,R0                           ; Are we in current extent?
                         09   19  03EA          BLSS    40$                             ; LSS means YES, in current.
              E3 55   5C A2   F2  03EC          AOBLSS  ADP$L_MRACTMDRS(R2),R5,20$      ; Loop thru all extents.
                                03F1  30$:                                            ; Failure if we fall thru.
                         50   D4  03F1          CLRL    R0                              ; Set failure code.
                         52   11  03F3          BRB     80$                             ; And branch to return.
                                03F5  40$:
                                03F5
                                03F5  ; Here the first register we want is greater than the first register of
```

```
                           03F5 ;       current extent (defined by R5 = index) and is less than or equal to
                           03F5 ;       the last register of the extent. R0 contains the # of the register just
                           03F5 ;       beyond the current extent.  In other words,
                           03F5 ;
                           03F5 ;       ADP$W_MRFREGARY(R2)[R5] < R4 < R0
                           03F5
             50   54   A2  03F5         SUBW    R4,R0                         ; R0 = length of subextent based at R4.
             53   50   B1  03F8         CMPW    R0,R3                         ; Compare to # of registers needed.
                  F4   19  03FB         BLSS    30$                           ; LSS means failure.
                           03FD
             61   54   B0  03FD         MOVW    R4,UBMD$W_MAPREG(R1)    ; Success. Fill in user's descriptor
          02 A1   53   90  0400         MOVB    R3,UBMD$B_NUMREG(R1)    ;  with base register and # of registers.
                           0404
                           0404 ;       SUBW3   ADP$W_MRFREGARY(R2)[R5],R4,-    ; Distance from beginning of
                           0404 ;               ADP$W_MRNREGARY(R2)[R5]         ;  extent to R4 is new length.
        64 A245   50   A2  0404         SUBW    R0,ADP$W_MRNREGARY(R2)[R5]      ; Equivalent result.
                           0409
             50   53   A2  0409         SUBW    R3,R0                         ; R0 = # regs. left in sub-extent.
                  36   13  040C         BEQL    70$                           ; EQL means we do not have to allocate
                           040E                                               ;  and fill a new extent descriptor.
                  55   D6  040E         INCL    R5                            ; R5 = index of new extent descriptor.
             7E   50   B0  0410         MOVW    R0,-(SP)                      ; Save length of new extent.
                0OC9   30  0413         BSBW    ALLOC_DESCRIP                 ; Call to allocate a new descriptor.
                           0416
 015E C245   53   54   A1  0416         ADDW3   R4,R3,ADP$W_MRFREGARY(R2)[R5]   ; Fill in new descriptor with
        64 A245   8E   B0  041D         MOVW    (SP)+,ADP$W_MRNREGARY(R2)[R5]   ;  1st register and # registers.
                  20   11  0422         BRB     70$                             ; Branch around to success.
                           0424 50$:
                           0424
                           0424 ; Here the first register we want is equal to the first register of the current
                           0424 ;       extent (defined by index register R5).  In other words,
                           0424 ;
                           0424 ;       R4 = ADP$W_MRFREGARY(R2)[R5]
                           0424
        64 A245   53   B1  0424         CMPW    R3,ADP$W_MRNREGARY(R2)[R5]      ; See if we have enough registers.
                  C6   14  0429         BGTR    30$                             ; GTR implies failure.
                           042B
             61   54   B0  042B         MOVW    R4,UBMD$W_MAPREG(R1)    ; Success.  Fill in user's descriptor
          02 A1   53   B0  042E         MOVW    R3,UBMD$B_NUMREG(R1)    ;  with 1st register and # allocated.
                           0432
        64 A245   53   A2  0432         SUBW    R3,ADP$W_MRNREGARY(R2)[R5]      ; Update current descriptor.
                  08   13  0437         BEQL    60$                             ; EQL means current extent now
                           0439                                                 ;  empty. Go to deallocate.
      015E C245   53   A0  0439         ADDW    R3,ADP$W_MRFREGARY(R2)[R5]      ; If not empty, update 1st register.
                  03   11  043F         BRB     70$                             ; Branch around deallocate.
                           0441 60$:
                0082   30  0441         BSBW    DEALLOC_DESCRIP                 ; Deallocate system descriptor.
             50   01   D0  0444 70$:    MOVL    S^#SS$_NORMAL,R0        ; Set success indicator.
                  55 8ED0  0447 80$:    POPL    R5                      ; Restore R5,R4,R3
             53   8E   7D  044A         MOVQ    (SP)+,R3                ;
                       05  044D         RSB                             ; And return to caller.
```

```
                                044E         .SBTTL  Permanently Allocate UNIBUS Map Registers
                                044E  ;+
                                044E  ; IOC$ALOUBAMAPRM  - Permanently Allocate UBA Map Registers (CRB Database Specified)
                                044E  ; IOC$ALOUBAMAPRMN - Permanently Allocate UBA Map Registers (Argument Specified)
                                044E  ;
                                044E  ; This routine is called to permanently allocate UNIBUS map registers.
                                044E  ;         Here we allocate the map registers from the highest numbered
                                044E  ;         available registers.
                                044E  ;
                                044E  ; INPUTS:
                                044E  ;         R3 =  # Registers to allocate (IOC$ALOUBAMAPRMN only)
                                044E  ;         R5 => UCB
                                044E  ;
                                044E  ; OUTPUTS:
                                044E  ;         R0 = Success indication
                                044E  ;
                                044E  ;-
                                044E
                                044E         .enabl  LSB
                                044E  IOC$ALOUBMAPRMN::                                   ;ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFI
             7E      53    7D   044E         MOVQ    R3,-(SP)                            ; Save R3,R4,R5
                     55    DD   0451         PUSHL   R5                                  ;
                                0453
                     18    11   0453         BRB     5$                                  ;
                                0455  IOC$ALOUBMAPRM::                                    ;ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
             7E      53    7D   0455         MOVQ    R3,-(SP)                            ; Save R3,R4,R5
                     55    DD   0458         PUSHL   R5                                  ;
                                045A
          53      7E A5    3C   045A         MOVZWL  UCB$W_BCNT(R5),R3                   ;GET TRANSFER BYTE COUNT
          54      7C A5    3C   045E         MOVZWL  UCB$W_BOFF(R5),R4                   ;GET BYTE OFFSET IN PAGE
       53   03FF C344      9E   0462         MOVAB   ^X3FF(R3)[R4],R3                    ;CALCULATE HIGHEST RELATIVE BYTE AND ROUND
    53    53      F7 8F    78   0468         ASHL    #-9,R3,R3                           ;CALCULATE NUMBER OF MAP REGISTERS REQUIRED
                                046D  5$:
          51      24 A5    D0   046D         MOVL    UCB$L_CRB(R5),R1                       ; R1 => CRB
          52      38 A1    D0   0471         MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2            ; R2 => ADP
          51      34 A1    9E   0475         MOVAB   CRB$L_INTD+VEC$W_MAPREG(R1),R1         ; R1 => UBMD.
                     0F    E0   0479         BBS     #VEC$V_MAPLOCK,-                       ; If SET, already permanently
                  38 61         047B                 UBMD$W_MAPREG(R1),30$                  ;  allocated, so branch around.
                                047D
                     53    D6   047D         INCL    R3                                  ; Round up request to next multiple
              53     01    8A   047F         BICB    #1,R3                               ;  of 2.
          55      5C A2    D0   0482         MOVL    ADP$L_MRACTMDRS(R2),R5              ; R5 = index beyond last MRD.
                     0A    13   0486         BEQL    15$                                 ; EQL implies no registers available.
                                0488  10$:
        62 A245      53    B1   0488         CMPW    R3,ADP$W_MRNREGARY-2(R2)[R5]              ; See if enough regs described here.
                     07    15   048D         BLEQ    20$                                       ; LEQ implies YES.
                                048F
                  F6 55    F5   048F         SOBGTR  R5,10$                                    ; Else branch back and continue
                                0492  15$:
                     50    D4   0492         CLRL    R0                                        ; If here, allocation failure.
                     22    11   0494         BRB     40$                                       ; Branch around to return.
                                0496  20$:
50   62 A245   015C C245   A1   0496         ADDW3   ADP$W_MRFREGARY-2(R2)[R5],-               ; Calculate register # beyond
                                049F                 ADP$W_MRNREGARY-2(R2)[R5],R0              ;  last extent.
             50      53    A2   049F         SUBW    R3,R0                                     ; We allocate from high end. R0
                                04A2                                                           ;  contains 1st reg. to alloc.
             61      50    B0   04A2         MOVW    R0,UBMD$W_MAPREG(R1)                      ; Record 1st register allocated.
```

```
     61    8000 8F     A8  04A5   1253          BISW    #VEC$M_MAPLOCK,UBMD$W_MAPREG(R1);  and mark it permanent.
       02 A1    53     90  04AA   1254          MOVB    R3,UBMD$B_NUMREG(R1)              ; Set # of map regs allocated.
     62 A245    53     A2  04AE   1255          SUBW    R3,ADP$W_MRNREGARY-2(R2)[R5]      ; Subtract out # regs allocated.
                0A     13  04B3   1256          BEQL    50$                               ; EQL implies descriptor not
                           04B5   1257                                                    ;  valid; branch to deallocate.
                           04B5   1258 30$:
           50   01     D0  04B5   1259          MOVL    S^#SS$_NORMAL,R0           ; Indicate success.
                           04B8   1260 40$:
                55   8ED0  04B8   1261          POPL    R5                         ; Restore R5,R4,R3
           53   8E     7D  04BB   1262          MOVQ    (SP)+,R3                   ;
                       05  04BE   1263          RSB
                           04BF   1264 50$:
                55     D7  04BF   1265          DECL    R5                         ; R5 = index of descriptor to dealloc.
              0002     30  04C1   1266          BSBW    DEALLOC_DESCRIP            ; Call to deallocate descriptor.
                EF     11  04C4   1267          BRB     30$                        ;  And branch back to return.
                           04C6   1268          .dsabl  lsb
```

```
                                04C6  1270 ;+
                                04C6  1271 ; DEALLOC_DESCRIP - Common internal subroutine called to deallocate
                                04C6  1272 ;        a UBA Map Register descriptor.
                                04C6  1273 ;
                                04C6  1274 ; INPUTS:
                                04C6  1275 ;        R2 => ADP
                                04C6  1276 ;        R5 =  index of descriptor to deallocate.
                                04C6  1277 ; OUTPUTS:
                                04C6  1278 ;        The UBA Map Allocation structures are updated by contracting
                                04C6  1279 ;                descriptors over the deallocated one.
                                04C6  1280 ;        Register R5 is modified.
                                04C6  1281 ;-
                                04C6  1282
                                04C6  1283 DEALLOC_DESCRIP:
                   5C A2   D7   04C6  1284         DECL    ADP$L_MRACTMDRS(R2)     ; Decrement # active descriptors.
                                04C9  1285 10$:
       64 A245   66 A245   B0   04C9  1286         MOVW    ADP$W_MRNREGARY+2(R2)[R5],-     ; Move data towards lower index
                                04D0  1287                 ADP$W_MRNREGARY(R2)[R5]         ;  to fill up hole.
   015E C245   0160 C245   B0   04D0  1288         MOVW    ADP$W_MRFREGARY+2(R2)[R5],-     ;
                                04D9  1289                 ADP$W_MRFREGARY(R2)[R5]         ;
             EB 55   5C A2   F2   04D9  1290         AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$      ; Loop thru rest of active MDRS.
                           05   04DE  1291         RSB
                                04DF  1292
                                04DF  1293 ;+
                                04DF  1294 ; ALLOC_DESCRIP - Common internal subroutine to allocate a UBA map register
                                04DF  1295 ;        descriptor in the middle of the range of descriptors.
                                04DF  1296 ;
                                04DF  1297 ; INPUTS:
                                04DF  1298 ;        R2 => ADP
                                04DF  1299 ;        R5 =  index of where we must allocate descriptor
                                04DF  1300 ; OUTPUTS:
                                04DF  1301 ;        Allocation is accomplished by creating a hole in each of the arrays
                                04DF  1302 ;                by moving descriptor items to the next higher element.
                                04DF  1303 ;        Note R0 is modified.
                                04DF  1304 ;-
                                04DF  1305
                                04DF  1306 ALLOC_DESCRIP:
              50   5C A2   D0   04DF  1307         MOVL    ADP$L_MRACTMDRS(R2),R0          ; R0 = # active descriptors.
                                04E3  1308 10$:
                   55   50   D1   04E3  1309         CMPL    R0,R5                           ; Have we finished?
                        13   15   04E6  1310         BLEQ    20$                             ; LEQ implies YES.
       64 A240   62 A240   B0   04E8  1311         MOVW    ADP$W_MRNREGARY-2(R2)[R0],-     ; Starting from ends of arrays,
                                04EF  1312                 ADP$W_MRNREGARY(R2)[R0]         ;  copy # register items.
                                04EF  1313
   015E C240   015C C240   B0   04EF  1314         MOVW    ADP$W_MRFREGARY-2(R2)[R0],-     ;
                                04F8  1315                 ADP$W_MRFREGARY(R2)[R0]         ;
                   E8 50   F5   04F8  1316         SOBGTR  R0,10$                          ; And loop back until we reach
                                04FB  1317                                                 ;  the hole we have created.
                   5C A2   D6   04FB  1318 20$:    INCL    ADP$L_MRACTMDRS(R2)             ; Increment # active descriptors.
                           05   04FE  1319         RSB                                     ; Return to caller
```

```
                          04FF           .SBTTL  Release UNIBUS Map Registers
                          04FF   ;+
                          04FF   ; IOC$RELMAPUDA - RELEASE UNIBUS MAP REGISTERS (CALLED FROM UDA PORT DRIVER)
                          04FF   ; IOC$RELMAPREG - RELEASE UNIBUS MAP REGISTERS
                          04FF   ;
                          04FF   ; This routine is called to release UNIBUS map registers that were previously
                          04FF   ; assigned for an I/O transfer.
                          04FF   ;
                          04FF   ; INPUTS:
                          04FF   ;   (For IOC$RELMAPUDA only)
                          04FF   ;
                          04FF   ;        R4 => PDT
                          04FF   ;        R5 => CDRP
                          04FF   ;
                          04FF   ;   (For IOC$RELMAPREG call only)
                          04FF   ;
                          04FF   ;        R5 = UCB ADDRESS OF DEVICE UNIT.
                          04FF   ;
                          04FF   ;        It is assumed that the caller still owns the I/O channel on which
                          04FF   ;        the transfer took place.
                          04FF   ;
                          04FF   ; OUTPUTS:
                          04FF   ;
                          04FF   ;        If the mapping registers have been permanently assigned to the asso-
                          04FF   ;        ciated I/O channel (only possible for IOC$RELMAPREG), then control
                          04FF   ;        is immediately returned to the caller.  Else the mapping registers are
                          04FF   ;        released (via a call to IOC$DALOCUBAMAP) and we then go into a loop
                          04FF   ;        removing waiting driver processes from the Map Register Wait Queue
                          04FF   ;        until either the Queue is completely drained or we run out of map
                          04FF   ;        registers to satisfy the needs of a given waiting driver process.
                          04FF   ;        Driver processes waiting here have their context stored in either
                          04FF   ;        a UCB fork block or a CDRP fork block and the processing required to
                          04FF   ;        resume each of these types of driver process is slightly different.
                          04FF   ;        What is done for each is to allocate the required map registers
                          04FF   ;        (via a call to IOC$ALOUBAMPA for UCB threads and via a call to
                          04FF   ;        IOC$ALOUBAMAP for CDRP threads) and to resume the waiting driver
                          04FF   ;        process.  Resuming a UCB thread is done by restoring register
                          04FF   ;        context and JSB'ing to the saved PC.  Resuming a CDRP thread is
                          04FF   ;        accomplished by calling SCS$RESUMEWAITR.
                          04FF   ;-
                          04FF           .enabl  lsb
                          04FF   IOC$RELMAPUDA::
        7E    53     7D   04FF           MOVQ    R3,-(SP)                ; Save R3-R6
        7E    55     7D   0502           MOVQ    R5,-(SP)                ;
                          0505
52    00E0    C4     D0   0505           MOVL    PDT$L_ADP(R4),R2        ; R2 => ADP.
        56    52     D0   050A           MOVL    R2,R6                   ; R6 => ADP also.
                          050D
  53     3C   A5     9E   050D           MOVAB   CDRP$L_UBARSRCE(R5),R3  ; R3 => UBMD.
     54       63     3C   0511           MOVZWL  UBMD$W_MAPREG(R3),R4    ; R4 has 1st mapreg #.
  53     02   A3     9A   0514           MOVZBL  UBMD$B_NUMREG(R3),R3    ; R3 has # of mapregs.
              1E     11   0518           BRB     10$                     ; Branch to common code.
                          051A
                          051A   IOC$RELMAPREG::                         ; Release unibus map registers
  51     24   A5     D0   051A           MOVL    UCB$L_CRB(R5),R1        ; R1 => CRB.
              0F     E0   051E           BBS     #VEC$V_MAPLOCK,-        ; If SET, permanent allocation so branch.
   3D 34   A1             0520                   CRB$L_INTD+VEC$W_MAPREG(R1),50$
```

```
         7E    53    7D  0523              MOVQ    R3,-(SP)                   ; Save R3-R6
         7E    55    7D  0526              MOVQ    R5,-(SP)                   ;
                         0529
      52    38 A1    D0  0529              MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2 ;GET ADDRESS OF ADP
         56    52    D0  052D              MOVL    R2,R6                      ;SAVE ADDRESS OF ADP
      54    34 A1    3C  0530              MOVZWL  CRB$L_INTD+VEC$W_MAPREG(R1),R4 ;GET STARTING MAP REGISTER NUMBER
      53    36 A1    9A  0534              MOVZBL  CRB$L_INTD+VEC$B_NUMREG(R1),R3 ;GET NUMBER OF REGISTERS TO DEALLOC
                         0538  10$:
             0038    30  0538              BSBW    IOC$DALOCUBAMAP            ; Free up UBA map resources.
                         053B  20$:
      55    30 B6    0F  053B              REMQUE  @ADP$L_MRQFL(R6),R5        ;GET ADDRESS OF NEXT DRIVER FORK BLOCK
               19    1D  053F              BVS     40$                        ;IF VS NO DRIVER PROCESS WAITING
                         0541
               10    91  0541              CMPB    #DYN$C_UCB,-               ; See if we dequeued a UCB or a CDRP.
            0A A5        0543              UCB$B_TYPE(R5)
               1A    12  0545              BNEQ    REALLOC_CD_MAPREGS         ; NEQ implies a CDRP.
                         0547
             FDFB    30  0547              BSBW    IOC$ALOUBAMAP              ;SEARCH MAP REGISTER BITMAP AND ALLOCATE
            09 50    E9  054A              BLBC    R0,30$                     ;IF LBC ALLOCATION FAILURE
      53    10 A5    7D  054D              MOVQ    UCB$L_FR3(R5),R3           ;RESTORE DRIVE PROCESS CONTEXT
            0C B5    16  0551              JSB     @UCB$L_FPC(R5)             ;CALL DRIVER AT MAP REGISTER WAIT RETURN ADD
               E5    11  0554              BRB     20$                        ;
   30 A6       65    0E  0556  30$:        INSQUE  UCB$L_FQFL(R5),ADP$L_MRQFL(R6) ;REINSERT DRIVER PROCESS AT FRONT OF
         55    8E    7D  055A  40$:        MOVQ    (SP)+,R5                   ; Restore R3-R6
         53    8E    7D  055D              MOVQ    (SP)+,R3                   ;
                     05  0560  50$:        RSB                                ;
                         0561
                         0561  REALLOC_CD_MAPREGS:                            ; Reallocate mapregs to a class driver
                         0561                                                 ;  process.
      54    14 A5    D0  0561              MOVL    CDRP$L_FR4(R5),R4          ; Restore saved fork register.
             FDB3    30  0565              BSBW    IOC$ALOMAPUDA              ; Allocate map registers if we can.
            EB 50    E9  0568              BLBC    R0,30$                     ; LBC implies allocation failure, branch
                         056B
      00000000'EF    16  056B              JSB     SCS$RESUMEWAITR            ; Resume waiting thread and any backed
                         0571                                                 ;  up IRP's.
               C8    11  0571              BRB     20$                        ; Branch back to try and allocate more
                         0573                                                 ;  UNIBUS map registers.
                         0573              .dsabl  lsb
```

```
                              0573  ;+
                              0573  ; IOC$DALOCUBAMAP - Common internal subroutine to update the UBA Map allocation
                              0573  ;         structures to include the map registers specified here among the
                              0573  ;         available map registers.
                              0573  ;
                              0573  ; INPUTS:
                              0573  ;         R2 => ADP
                              0573  ;         R3 =  # map registers to free.
                              0573  ;         R4 =  first map register to free.
                              0573  ;
                              0573  ; OUTPUTS:
                              0573  ;         The UBA Map Allocation structures are updated.
                              0573  ;
                              0573  ;         Registers R0, R1 and R5 are modified.
                              0573  ;
                              0573  ;-
                              0573
                              0573  IOC$DALOCUBAMAP:
                     55   D4  0573          CLRL    R5                          ; Initialize loop variable.
            51   53  54   C1  0575          ADDL3   R4,R3,R1                    ; R1 = map register beyond extent.
                     53   D5  0579          TSTL    R3                          ; Is the # of regs. to deallocate zero?
                     62   13  057B          BEQL    90$                         ; Branch to bugcheck if zero.
                  5C A2   D5  057D          TSTL    ADP$L_MRACTMDRS(R2)         ; Test for zero active descriptors.
                     4E   13  0580          BEQL    50$                         ; EQL implies no registers available.
                              0582  10$:
       015E C245     51   B1  0582          CMPW    R1,ADP$W_MRFREGARY(R2)[R5]        ; See if map registers to free
                              0588                                                     ;   are before those described
                              0588                                                     ;   by current descriptor.
                     07   15  0588          BLEQ    20$                               ; LEQ implies yes.
                              058A
        F3 55     5C A2   F2  058A          AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$        ; Else branch back and try next.
                     28   11  058F          BRB     40$                               ; If here, registers to free
                              0591                                                     ;  beyond those described by
                              0591                                                     ;  last descriptor. So branch
                              0591                                                     ;  to try and absorb at end of
                              0591                                                     ;  last descriptor.
                              0591  20$:
                     29   12  0591          BNEQ    40$                         ; NEQ implies that although we alloca-
                              0593                                              ;  registers before the current des-
                              0593                                              ;  criptor, we are not contiguous with
                              0593                                              ;  it.  So we branch to try and absorb
                              0593                                              ;  these registers in the previous one.
                              0593
                              0593  ; Here we can absorb the registers in the current descriptor.
                              0593
 50   015C C245   62 A245   A1  0593          ADDW3   ADP$W_MRNREGARY-2(R2)[R5],-      ; Calculate end of previous
                              059C                  ADP$W_MRFREGARY-2(R2)[R5],R0     ;  extent and move to R0.
                  54     50   B1  059C          CMPW    R0,R4                            ; Does it coincide with start
                              059F                                                      ;  of this extent?
                     0C   13  059F          BEQL    30$                              ; EQL implies yes.
                              05A1
                              05A1  ; Here we have the most likely case. The map registers that we are freeing can
                              05A1  ;         be absorbed into the top of the current descriptor but not also in the
                              05A1  ;         previous descriptor.
                              05A1
       015E C245     54   B0  05A1          MOVW    R4,ADP$W_MRFREGARY(R2)[R5]        ; First register freed becomes
                              05A7                                                     ;  first register of current
```

```
                                  05A7                                                       ;  descriptor.
          64 A245    53   A0     05A7             ADDW    R3,ADP$W_MRNREGARY(R2)[R5]         ; Number of registers is sum of
                                  05AC                                                       ;  registers freed and registers
                                  05AC                                                       ;  previously described here.
                          05     05AC             RSB
                                  05AD
                                  05AD     ; Here we have the case where the map registers being freed fall between two
                                  05AD     ;         discontiguous blocks and exactly span the difference.  We then can
                                  05AD     ;         describe the entire group with one descriptor, and so we also
                                  05AD     ;         deallocate the current descriptor.  Note new combined descriptor
                                  05AD     ;         will still begin at same map register number so we do NOT alter
                                  05AD     ;         this item.
                                  05AD
                                  05AD     30$:
          62 A245    53   A0     05AD             ADDW    R3,ADP$W_MRNREGARY-2(R2)[R5]       ; Partial sum of registers
                                  05B2                                                       ;  being freed and previous ones.
     62 A245    64 A245   A0     05B2             ADDW    ADP$W_MRNREGARY(R2)[R5],-          ; Now add in registers described
                                  05B9                    ADP$W_MRNREGARY-2(R2)[R5]          ;  in current descriptor.
                                  05B9
                   FF0A   31     05B9             BRW     DEALLOC_DESCRIP                    ; BRW to subroutine and let it
                                  05BC                                                       ;  return to our caller.
                                  05BC
                                  05BC     ; Here we cannot absorb the freed map registers in the current descriptor.
                                  05BC     ;         We test to see if we can absorb them in the previous descriptor.
                                  05BC
                                  05BC     40$:
 50    015C C245    62 A245   A1     05BC             ADDW3   ADP$W_MRNREGARY-2(R2)[R5],-        ; Calculate end of previous
                                  05C5                    ADP$W_MRFREGARY-2(R2)[R5],R0       ;  extent and move to R0.
                 54    50   B1     05C5             CMPW    R0,R4                              ; See if contigous with previous.
                       06   12     05C8             BNEQ    50$                                ; NEQ implies NO.
                                  05CA
          62 A245    53   A0     05CA             ADDW    R3,ADP$W_MRNREGARY-2(R2)[R5]       ; Sum # of registers in extent.
                          05     05CF             RSB
                                  05D0
                                  05D0     ; Here we must allocate a new descriptor to describe the map registers we
                                  05D0     ;         are freeing.  Conditions at this time are as follows:
                                  05D0     ;
                                  05D0     ;                        R2 => ADP
                                  05D0     ;                        R3 =  # registers to free
                                  05D0     ;                        R4 =  first register to free
                                  05D0     ;                        R5 =  index of where we must allocate descriptor
                                  05D0     ;
                                  05D0     ;         Allocation is accomplished by calling subroutine ALLOC_DESCRIP
                                  05D0     ;
                                  05D0
                                  05D0     50$:
                   FF0C   30     05D0             BSBW    ALLOC_DESCRIP                      ; Alloc R5 = index of descriptor.
          64 A245    53   B0     05D3             MOVW    R3,ADP$W_MRNREGARY(R2)[R5]         ; Fill in allocated descriptor.
        015E C245    54   B0     05D8             MOVW    R4,ADP$W_MRFREGARY(R2)[R5]         ;
                          05     05DE             RSB
                                  05DF
                                  05DF     90$:   BUG_CHECK INCONSTATE               ; Non-fatal bugcheck on zero map
                                  05E3                                               ; registers deallocation attempts.
                          05     05E3             RSB                                ; Then ignore deallocate request.
        05E4               .SBTTL  RETURN TO CALLER
        05E4       ;+
        05E4       ; IOC$RETURN - RETURN TO CALLER
        05E4       ;
        05E4       ; THIS ROUTINE IS CALLED AS A RESULT OF A DDT DISPATCH TO A NULL ENTRY. ITS
        05E4       ; FUNCTION IS MERELY TO RETURN TO ITS CALLER.
        05E4       ;
        05E4       ; INPUTS:
        05E4       ;
        05E4       ;       NONE.
        05E4       ;
        05E4       ; OUTPUTS:
        05E4       ;
        05E4       ;       NONE.
        05E4       ;-
        05E4
        05E4       IOC$RETURN::                          ;RETURN TO CALLER
    05  05E4               RSB                           ;
```

```
                                05E5         .SBTTL  WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
                                05E5   ;+
                                05E5   ; IOC$WFIKPCH - WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
                                05E5   ;
                                05E5   ; THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON
                                05E5   ; A DEVICE UNIT AND TO KEEP THE CHANNEL. THIS ROUTINE CAN BE CALLED AT
                                05E5   ; EITHER FORK OR DEVICE INTERRUPT LEVEL.
                                05E5   ;
                                05E5   ; INPUTS:
                                05E5   ;
                                05E5   ;       00(SP) = RETURN ADDRESS OF CALLER.
                                05E5   ;       04(SP) = TIMEOUT VALUE IN SECONDS.
                                05E5   ;       08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
                                05E5   ;       12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                                05E5   ;
                                05E5   ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                                05E5   ;
                                05E5   ; OUTPUTS:
                                05E5   ;
                                05E5   ;       THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
                                05E5   ;       R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
                                05E5   ;       TIMEOUT ARE ENABLED, AND A RETURN TO THE CALLER'S CALLER IS EXECUTED.
                                05E5   ;-
                                05E5
                                05E5   IOC$WFIKPCH::                                   ;WAITFOR INTERRUPT/TIMEOUT AND KEEP CHANNEL
              6E   02     C0    05E5           ADDL    #2,(SP)                         ;CALCULATE OFFSET TO NORMAL RETURN
          10 A5    53     7D    05E8           MOVQ    R3,UCB$L_FR3(R5)                ;SAVE REGISTERS R3 AND R4
             0C A5 8ED0         05EC           POPL    UCB$L_FPC(R5)                   ;SAVE INTERRUPT RETURN ADDRESS
          64 A5    03     A8    05F0           BISW    #UCB$M_INT!UCB$M_TIM,UCB$W_STS(R5) ;ENABLE INTERRUPT AND TIMEOUT
6C A5  00000000'EF  8E    C1    05F4           ADDL3   (SP)+,G^EXE$GL_ABSTIM,UCB$L_DUETIM(R5) ;SET TIMEOUT TIME
    64 A5    0040 8F      AA    05FD           BICW    #UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR UNIT TIMED OUT
                                0603           ENBINT                                  ;ENABLE INTERRUPTS
                          05    0606           RSB                                     ;
```

```
                                   0607  1582          .SBTTL  WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
                                   0607  1583  ;+
                                   0607  1584  ; IOC$WFIRLCH - WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
                                   0607  1585  ;
                                   0607  1586  ; THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON A DEVICE
                                   0607  1587  ; UNIT AND TO RELEASE THE CHANNEL. THIS ROUTINE CAN ONLY BE CALLED AT FORK LEVEL.
                                   0607  1588  ;
                                   0607  1589  ; INPUTS:
                                   0607  1590  ;
                                   0607  1591  ;       00(SP) = RETURN ADDRESS OF CALLER.
                                   0607  1592  ;       04(SP) = TIMEOUT VALUE IN SECONDS.
                                   0607  1593  ;       08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
                                   0607  1594  ;       12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                                   0607  1595  ;
                                   0607  1596  ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                                   0607  1597  ;
                                   0607  1598  ; OUTPUTS:
                                   0607  1599  ;
                                   0607  1600  ;       THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
                                   0607  1601  ;       R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
                                   0607  1602  ;       TIMEOUT ARE ENABLED, THE CHANNEL IS RELEASED, AND A RETURN TO THE CALLER'S
                                   0607  1603  ;       CALLER IS EXECUTED.
                                   0607  1604  ;-
                                   0607  1605
                                   0607  1606  IOC$WFIRLCH::                                   ;WAITFOR INTERRUPT/TIMEOUT AND RELEASE CHANN
                 6E     02     C0  0607  1607          ADDL    #2,(SP)                         ;CALCULATE OFFSET TO NORMAL RETURN
              10 A5     53     7D  060A  1608          MOVQ    R3,UCB$L_FR3(R5)                ;SAVE REGISTERS R3 AND R4
                 0C     A5   8ED0  060E  1609          POPL    UCB$L_FPC(R5)                   ;SAVE INTERRUPT RETURN ADDRESS
              64 A5     03     A8  0612  1610          BISW    #UCB$M_INT!UCB$M_TIM,UCB$W_STS(R5) ;ENABLE INTERRUPT AND TIMEOUT
   6C A5   00000000'EF  8E     C1  0616  1611          ADDL3   (SP)+,L^EXE$GL_ABSTIM,UCB$L_DUETIM(R5) ;SET TIMEOUT TIME
        64 A5     0040  8F     AA  061F  1612          BICW    #UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR UNIT TIMED OUT
                                   0625  1613          ENBINT                                  ;ENABLE INTERRUPTS
                    FA5F       31  0628  1614          BRW     IOC$RELCHAN                     ;RELEASE ALL CHANNELS AND RETURN TO CALLER
                                   062B  1615
                                   062B  1616
```

```
                                 062B   1618          .SBTTL  ALLOCATE SYSTEM PAGE TABLE
                                 062B   1619  ;+
                                 062B   1620  ; IOC$ALLOSPT - ALLOCATE SYSTEM PAGE TABLE
                                 062B   1621  ;
                                 062B   1622  ; THIS ROUTINE ALLOCATES SYSTEM PAGE TABLE (SPT) ENTRIES.
                                 062B   1623  ;
                                 062B   1624  ; INPUTS:
                                 062B   1625  ;
                                 062B   1626  ;       R1 = NUMBER OF SPT ENTRIES TO BE ALLOCATED
                                 062B   1627  ;
                                 062B   1628  ;       BOO$GL_SPTFREL = LOWEST FREE VPN
                                 062B   1629  ;       BOO$GL_SPTFREH = HIGHEST FREE VPN
                                 062B   1630  ;
                                 062B   1631  ;       IT IS ASSUMED THAT THE CALLER IS RUNNING AT IPL$_SYNCH.
                                 062B   1632  ;
                                 062B   1633  ; OUTPUTS:
                                 062B   1634  ;
                                 062B   1635  ;       R0 = SUCCESS INDICATION.
                                 062B   1636  ;       R2 = STARTING PAGE NUMBER ALLOCATED (SVPN).
                                 062B   1637  ;       R3 = ADDRESS OF BASE OF SYSTEM PAGE TABLE (MMG$GL_SPTBASE).
                                 062B   1638  ;
                                 062B   1639  ;       R1 IS PRESERVED ACROSS CALL.
                                 062B   1640  ;-
                                 062B   1641  IOC$ALLOSPT::                                   ;ALLOCATE SYSTEM PAGE TABLE
                    50     D4    062B   1642          CLRL    R0                              ;ASSUME FAILURE
 52    00000000'EF         D0    062D   1643          MOVL    L^BOO$GL_SPTFREL,R2             ;GET NEXT AVAILABLE SYSTEM VPN
       53     52    51     C1    0634   1644          ADDL3   R1,R2,R3                        ;COMPUTE NEXT WITH THIS ALLOCATION
 00000000'EF        53     D1    0638   1645          CMPL    R3,L^BOO$GL_SPTFREH             ;ARE THERE ENOUGH AVAILABLE?
                    10     1E    063F   1646          BGEQU   10$                             ;BR IF NO
 00000000'EF        53     D0    0641   1647          MOVL    R3,L^BOO$GL_SPTFREL             ;MARK THE ENTRIES ALLOCATED
 53    00000000'EF         D0    0648   1648          MOVL    L^MMG$GL_SPTBASE,R3             ;GET ADDR OF BASE OF SPT
                    50     D6    064F   1649          INCL    R0                              ;SET SUCCESS
                                 0651   1650  10$:
                           05    0651   1651          RSB                                     ;
```

```
0652          .SBTTL  CONVERT DEVICE NAME AND UNIT
0652   ;+
0652   ; IOC$CVT_DEVNAM - Convert device name and unit
0652   ;
0652   ; This routine is called to convert a device name and unit number to a physical
0652   ; device name string.
0652   ;
0652   ; Inputs:
0652   ;
0652   ;       The caller is assumed to have PROBEd the output buffer for write access.
0652   ;       The I/O data base is locked for read access.  This means that the caller
0652   ;       owns the I/O data base mutex and/or is at IPL SYNCH or higher.
0652   ;
0652   ;       R0 = length of output buffer.
0652   ;       R1 = address of output buffer.
0652   ;       R4 = name string formation mode, one of:
0652   ;           -1 (DVI$_DEVNAM) -- a name suitable for displays
0652   ;               for non-local devices, return node$ddcn
0652   ;               for local devices:
0652   ;                   if in cluster and file oriented device, return node$ddcn
0652   ;                   otherwise, return ddcn
0652   ;            0 (DVI$_FULLDEVNAM) -- a name with appropriate node information
0652   ;               if allocation class not zero and file oriented device, return
0652   ;                       $alloclass$ddcn
0652   ;               otherwise, return node$ddcn
0652   ;            1 (DVI$_ALLDEVNAM) -- a name with allocation class information
0652   ;               if allocation class not zero, return $alloclass$ddcn
0652   ;               otherwise, return node$ddcn
0652   ;            2 (no GETDVI item code) -- an old fashioned name
0652   ;               return ddcn
0652   ;            3 (no GETDVI item code) -- a secondary path name for displays
0652   ;               same as -1 except secondary path name returned
0652   ;            4 (no GETDVI item code) -- path controller name for displays
0652   ;               same as -1 except no unit number is appended
0652   ;            Note: if the node name string is null, node$ is not returned.
0652   ;       R5 = address of device UCB.
0652   ;
0652   ; Outputs:
0652   ;
0652   ;       The device name and unit number are converted and stored in the specified
0652   ;       output buffer. The following register values are returned:
0652   ;
0652   ;               R0 = Final conversion status.
0652   ;                       SS$_NORMAL or
0652   ;                       SS$_BUFFEROVF (an alternate success status which
0652   ;                               indicates that the supplied buffer could not
0652   ;                               hold the device name string)
0652   ;               R1 = Length of conversion string.  R1 = 0 if the alternate
0652   ;                       path name was requested but none exists.
0652   ;-
0652
0652   ;
0652   ; Working storage (offsets from R7)
0652   ;
0652          $OFFSET 0,POSITIVE,< -
0652                  <BINNUM,8>, -           ;Binary value to convert to ASCII
0652                  -       ; add new working storage cells before this line
```

```
                        0652                         <RESR0>, -              ;Result R0
                        0652                         <RESR1>, -              ;Result R1
                        0652                         <SCRLEN,0> -            ;amount of working storage
                        0652                         <RESR2>, -              ;saved R2
                        0652                         <RESR3>, -              ;saved R3
                        0652                         <RESR4>, -              ;saved R4
                        0652                         >
                        0000      BINNUM:
                        0008      RESR0:
                        000C      RESR1:
                        0010      SCRLEN:
                        0010      RESR2:
                        0014      RESR3:
                        0018      RESR4:
                        0652
                        0652      IOC$CVT_DEVNAM::                           ;Convert device name and unit
                        0652
        00FC 8F    BB   0652              PUSHR   #^M<R2,R3,R4,R5,R6,R7>  ;Save registers
                        0656      ;
                        0656      ; Push a quadword onto the stack.  The quadword will land
                        0656      ; on the stack so that when the POPR at the end of the routine
                        0656      ; is executed, R0 will contain the routine value, and R1 will
                        0656      ; contain the length of the formatted device name.
                        0656      ;
          7E   01  7D   0656              MOVQ    #SS$_NORMAL,-(SP)       ;Put a 1 and a 0 on the stack
               7E  7C   0659              CLRQ    -(SP)                   ;Init binary number working area.
                        065B              ASSUME  SCRLEN EQ 16
          57   5E  D0   065B              MOVL    SP, R7                  ;Setup result R0 and R1 pointer in R7.
                        065E      ;
                        065E      ; Precede the device name with a "_" (underscore character) to
                        065E      ; indicate that this is a physical device name.
                        065E      ;
          53   5F 8F 9A 065E              MOVZBL  #^A/_/,R3               ;Put underscore character in R3
             00B4  30   0662              BSBW    PUTCHAR                 ;Put it in the output buffer
                        0665      ;
                        0665      ; Check for a possible nodename. If it exists, determine which format
                        0665      ; of name was requested by the caller.
                        0665      ;
       56   28 A5  D0   0665              MOVL    UCB$L_DDB(R5),R6        ;Get DDB address
       52   34 A6  D0   0669              MOVL    DDB$L_SB(R6),R2         ;Get System Block address
               5D  13   066D              BEQL    LOCAL_NAME              ;None, leave
               09  E1   066F              BBC     #DEV$V_NNM,-            ;Branch if nodename not wanted
          58 3C A5      0671                      UCB$L_DEVCHAR2(R5),LOCAL_NAME
                        0674              CASE    R4, -                   ;Dispatch on type of output requested:
                        0674                      limit=#-1, displist=< -
                        0674                      DISPLAY_NAME, -         ; -1 ==> node$dev: for disks, else dev:
                        0674                      FULL_NAME, -            ;  0 ==> $allocls$dev: or node$dev:
                        0674                      ALLOC_NAME, -           ;  1 ==> $allocls$dev: or node$dev:
                        0674                      LOCAL_NAME, -           ;  2 ==> just dev:
                        0674                      SECONDARY_NAME, -       ;  3 ==> secondary path
                        0674                      DISPLAY_NAME -          ;  4 ==> same as -1 sans unit number
                        0674                      >
               5B  11   0686              BRB     EXDVNM                  ; All others are NOPs.
                        0688
                        0688      FULL_NAME:
    33 38 A5   0E  E1   0688              BBC     #DEV$V_FOD, -           ;A file oriented device?
                        068D                      UCB$L_DEVCHAR(R5), -
```

```
                          068D                 ADD_NODE                ;Branch if not file oriented device.
                          068D
                          068D ALLOC_NAME:
                          068D
        67    3C A6    9A 068D         MOVZBL  DDB$L_ALLOCLS(R6), -    ;Setup allocation class value
                          0691                 BINNUM(R7)              ; for conversion.
                 2D    13 0691         BEQL    ADD_NODE                ;If none return node+device name.
               0080    30 0693         BSBW    PUTDOLLAR               ;Prepend allocation class with a '$'
                 58    10 0696         BSBB    PUTNUM                  ;Convert allocation class number to
                          0698                                         ;ASCII and put it in the buffer
                 30    11 0698         BRB     ADD_DOLLAR              ;Append dollar sign to alloc. class
                          069A                                         ; and add device name to buffer.
                          069A
                          069A SECONDARY_NAME:
                 04    E1 069A         BBC     #DEV$V_2P,-             ;Branch if device not dual-pathed.
              3C A5       069C                 UCB$L_DEVCHAR2(R5),-    ; (I.E. there is no secondary path to
                 4C       069E                 NO_SECONDARY            ; return.)
        56  00A0 C5    D0 069F         MOVL    UCB$L_DP_DDB(R5),R6     ;Get secondary DDB.
                 45    13 06A4         BEQL    NO_SECONDARY            ;Branch to no sec. path if none.
        52    34 A6    D0 06A6         MOVL    DDB$L_SB(R6),R2         ;Get alternate SB.
                          06AA
                          06AA DISPLAY_NAME:
 00000000'8F     52    D1 06AA         CMPL    R2,#SCS$GA_LOCALSB      ;Is it the perm local system block?
                 0D    12 06B1         BNEQ    ADD_NODE                ;Return node+devnam for non-local devs.
                          06B3         IFNOCLSTR LOCAL_NAME            ;Return devnam if not part of a cluster.
   0C 38 A5      0E    E1 06BB         BBC     #DEV$V_FOD, -           ;A file oriented device?
                          06C0                 UCB$L_DEVCHAR(R5), -
                          06C0                 LOCAL_NAME              ;Branch if not a file oriented device.
                          06C0                                         ;Its a local disk in a cluster: return
                          06C0                                         ;node+device name format.
                          06C0 ;
                          06C0 ; Return node name plus device name.  Copy node name to buffer and
                          06C0 ; suffix with a "$" before moving in rest of device name.
                          06C0 ;
                          06C0 ADD_NODE:
        52    44 A2    9E 06C0         MOVAB   SB$T_NODENAME(R2),R2    ;Point to name field
                 62    95 06C4         TSTB    (R2)                    ;Is the node name null?
                 04    13 06C6         BEQL    LOCAL_NAME              ;Skip inserting node name, if its null.
                 3E    10 06C8         BSBB    PUTASCIC                ;Copy counted ASCII str. to output buf.
                          06CA ADD_DOLLAR:
                 4A    10 06CA         BSBB    PUTDOLLAR               ;Append dollar sign to node name
                          06CC ;
                          06CC ; Copy device name to buffer.
                          06CC ;
                          06CC LOCAL_NAME:
        52    14 A6    9E 06CC         MOVAB   DDB$T_NAME(R6),R2       ;Get address of ASCIC device name.
                 36    10 06D0         BSBB    PUTASCIC                ;Copy counted ASCII str. to output buf.
        04    18 A7    B1 06D2         CMPW    RESR4(R7),#4            ;Do we want the unit number?
                 0B    13 06D6         BEQL    EXDVNM                  ;Nope
        67    54 A5    3C 06D8         MOVZWL  UCB$W_UNIT(R5), -       ;Setup device unit number for
                          06DC                 BINNUM(R7)              ; converstion to ASCII.
                 12    10 06DC         BSBB    PUTNUM                  ;Convert unit number to ASCII.
                          06DE ;
                          06DE ; Terminate the device name with a ":" (colon).
                          06DE ;
           53    3A    9A 06DE         MOVZBL  #^A/:/,R3               ;Put a ":" in R3
                 36    10 06E1         BSBB    PUTCHAR                 ;Put the ":" in output buffer
```

```
                          06E3  ;
                          06E3  ; Clean up the stack and exit.  The stack has been set up so that
                          06E3  ; the proper values will be stored in R0 and R1 by the POPR.
                          06E3  ;
          5E    08    C0  06E3  EXDVNM: ADDL    #RESR0,SP                       ;Remove everything upto result R0
                          06E6                                                  ;from the stack
        00FF 8F       BA  06E6          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7> ;Restore registers
                      05  06EA          RSB                                     ;Return
                          06EB
                          06EB  ;
                          06EB  ; Come here when the secondary device name was requested but none exists.
                          06EB  ;
                          06EB  NO_SECONDARY:
             0C A7    D4  06EB          CLRL    RESR1(R7)                       ;Clear count of characters
                F3    11  06EE          BRB     EXDVNM                          ;and return.
                          06F0
                          06F0
                          06F0  ;++
                          06F0  ; The following code is a local subroutine to convert binary to ASCII and
                          06F0  ; put the ASCII equivalent in the output name buffer.
                          06F0  ;
                          06F0  ; Inputs:
                          06F0  ;
                          06F0  ;       BINNUM(R7)      binary number to be converted (a quadword with high
                          06F0  ;                       longword zeroed
                          06F0  ;
                          06F0  ; Outputs:
                          06F0  ;       The number at BINNUM(R7) is converted to ASCII and stored in the
                          06F0  ;       device name buffer.
                          06F0  ;--
                          06F0  PUTNUM:
          53    01    8E  06F0          MNEGB   #1, R3                          ;Get end-of-number marker.
          7E    53    90  06F3  10$:    MOVB    R3, -(SP)                       ;Move digit/marker to scratch.
53   67   67    0A    7B  06F6          EDIV    #10, BINNUM(R7), -              ;Divide number by 10, overwrite number
                          06FB                  BINNUM(R7), R3                  ;with quotient, put remainder in R3.
                F6    12  06FB          BNEQ    10$                             ;If quotient not zero, go save this
                          06FD                                                  ; digit and get the next one.
                          06FD  ;
                          06FD  ; Get digits -- most significant first (then saved ones), convert them to
                          06FD  ; ASCII, and put them in the output buffer
                          06FD  ;
          53    30    80  06FD  50$:    ADDB    #^A/0/, R3                      ;Convert binary digit to ASCII
                17    10  0700          BSBB    PUTCHAR                         ;Copy digit to output buffer
          53    8E    90  0702          MOVB    (SP)+, R3                       ;Get another digit
                F6    18  0705          BGEQ    50$                             ;Branch if the end
                      05  0707          RSB
                          0708
                          0708  ;++
                          0708  ; The following code is a local subroutine to copy a counted ASCII string
                          0708  ; to the output name buffer.
                          0708  ;
                          0708  ; Inputs:
                          0708  ;
                          0708  ;       R2      Beginning address of a counted ASCII string
                          0708  ;
                          0708  ; Outputs:
                          0708  ;       The counted ASCII string pointed to by R2 is copied to the device
```

```
                              0708   1874 ;       name buffer.
                              0708   1875 ;--
                              0708   1876 PUTASCIC:
              54   82   9A    0708   1877         MOVZBL  (R2)+, R4               ;Get counted string length.
                   08   13    070B   1878         BEQL    90$                     ;If no characters, leave.
              53   82   90    070D   1879 5$:     MOVB    (R2)+, R3               ;Move one byte to output buffer.
                   07   10    0710   1880         BSBB    PUTCHAR                 ;Put the character in the output buffer.
                F8 54   F5    0712   1881         SOBGTR  R4, 5$                  ;Branch if more to copy.
                        05    0715   1882 90$:    RSB                             ;All done, return.
                              0716   1883
                              0716   1884 ;++
                              0716   1885 ;
                              0716   1886 ; The following code is a local subroutine to place a given
                              0716   1887 ; byte in the output buffer.  A count is kept of all characters
                              0716   1888 ; placed in the output buffer.  If the output buffer is full,
                              0716   1889 ; the byte is not copied, the count is not increased, and the
                              0716   1890 ; return status for IOC$CVT_DEVNAM is changed to SS$_BUFFEROVF
                              0716   1891 ; (an alternate success status).
                              0716   1892 ;
                              0716   1893 ; Inputs:
                              0716   1894 ;       R0      Count of unstored character slots remaining in output buffer
                              0716   1895 ;       R1      Address of next unused character slot in output buffer
                              0716   1896 ;       R3      Character to be placed in the buffer
                              0716   1897 ;
                              0716   1898 ; Implicit inputs:
                              0716   1899 ;       RESR0(R7)       longword holding final IOC$CVT_DEVNAM status
                              0716   1900 ;       RESR1(R7)       longword holding final IOC$CVT_DEVNAM count of
                              0716   1901 ;                               characters stored in the buffer (to be
                              0716   1902 ;                               returned in R1
                              0716   1903 ;
                              0716   1904 ; Outputs:
                              0716   1905 ;       None.
                              0716   1906 ;
                              0716   1907 ; Implicit outputs:
                              0716   1908 ;       If R0 >= zero:
                              0716   1909 ;               R0              <== R0 - 1
                              0716   1910 ;               (R1)            <== R3
                              0716   1911 ;               R1              <== R1 + 1
                              0716   1912 ;               RESR1(R7)       <== RESR1(R7) + 1
                              0716   1913 ;       otherwise:
                              0716   1914 ;               RESR0(R7)       <== SS$_BUFFEROVF
                              0716   1915 ;++
                              0716   1916 ; PUTDOLLAR is an internal routine which is the equivalent of:
                              0716   1917 ;
                              0716   1918 ;       MOVB    #^A/$/, R3
                              0716   1919 ;       BSBB    PUTCHAR
                              0716   1920 ;--
                              0716   1921 PUTDOLLAR:
              53   24   90    0716   1922         MOVB    #^A/$/, R3              ;Setup to put "$" in output buffer.
                              0719   1923 PUTCHAR:
                   50   D7    0719   1924         DECL    R0                      ;Decrease characters remaining count.
                   07   19    071B   1925         BLSS    90$                     ;Branch if no more characters remaining.
              81   53   90    071D   1926         MOVB    R3, (R1)+               ;Copy character to output buffer
                0C A7   D6    0720   1927         INCL    RESR1(R7)               ;Count characters stored
                        05    0723   1928         RSB                             ;Return
                              0724   1929
   08 A7    0601 8F     3C    0724   1930 90$:    MOVZWL  #SS$_BUFFEROVF, -       ;Set buffer overflow status
```

```
    072A  1931              RESRO(R7)
 05 072A  1932          RSB
```

```
                        072B        .SBTTL  BROADCAST TO A TERMINAL
                        072B  ;++
                        072B  ; IOC$BROADCAST
                        072B  ;
                        072B  ;       This routine will allow driver fork processes to broadcast a
                        072B  ;       given message to given terminal.  The broadcast request is
                        072B  ;       dispatched to the proper terminal and control returns immediately
                        072B  ;       to the caller.  Some time later the broadcast will complete, and
                        072B  ;       at that time all the necessary post-processing will be done.
                        072B  ;
                        072B  ;       This routine does not implement all the features of the $BRDCST system
                        072B  ;       service, but only the bare minimum necessary to send a message to a
                        072B  ;       single terminal.  For more information about the terminal broadcast
                        072B  ;       mechanism, see the module SYSBRDCST.
                        072B  ;
                        072B  ; Input:
                        072B  ;
                        072B  ;       R1 = Message length
                        072B  ;       R2 = Message address
                        072B  ;       R5 = Address of target terminal's UCB
                        072B  ;
                        072B  ; Implicit input:
                        072B  ;
                        072B  ;       IPL$_ASTDEL  <=  CURRENT_IPL  <=  UCB$B_FIPL(R5)
                        072B  ;
                        072B  ; Output:
                        072B  ;
                        072B  ;       None.  The contents of R1 .. R5 are preserved across the call.
                        072B  ;
                        072B  ; Routine value:
                        072B  ;
                        072B  ;       SS$_NORMAL      - The broadcast completed successfully.
                        072B  ;       SS$_INSFMEM     - Insufficient dynamic nonpaged pool for the request.
                        072B  ;       SS$_DEVOFFLINE  - The target terminal has rejected the request.
                        072B  ;       SS$_ILLIOFUNC   - The specified UCB does not belong to a terminal.
                        072B  ;                         (Therefore a BROADCAST is an illegal I/O function.)
                        072B  ;--
                        072B
              00000000  072B  SAVED_R0 = 0                                    ;.
              00000004  072B  SAVED_R1 = 4                                    ; .
              00000008  072B  SAVED_R2 = 8                                    ; .
              0000000C  072B  SAVED_R3 = 12                                   ; . Symbolic offsets to saved registers
              00000010  072B  SAVED_R4 = 16                                   ; .
              00000014  072B  SAVED_R5 = 20                                   ;.
                        072B
                        072B  IOC$BROADCAST::                                 ; Broadcast to a terminal
 50   00F4 8F     3C    072B          MOVZWL  #SS$_ILLIOFUNC,R0               ; Assume device not a terminal
             02   E1    0730          BBC     #DEV$V_TRM,-                    ; Branch if not a terminal
       56 38 A5         0732                  UCB$L_DEVCHAR(R5),14$           ;
             3F   BB    0735          PUSHR   #^M<R0,R1,R2,R3,R4,R5>          ; Save R0 .. R5
       51    30   C0    0737          ADDL2   #TTY$K_WB_LENGTH,R1             ; Calculate the total pool required
 6E   0124 8F     3C    073A          MOVZWL  #SS$_INSFMEM,SAVED_R0(SP); Assume allocation failure
          F8BE'   30    073F          BSBW    EXE$ALONONPAGED                 ; Allocate the pool
          44 50   E9    0742          BLBC    R0,13$                          ; Exit if error
                        0745          ;
                        0745          ; Fill in the Terminal Write Packet (TWP).
                        0745          ; Note that EXE$ALONONPAGED the pool size
```

```
                    0745             ; in R1 and the pool address in R2.
                    0745             ;
08 A2    51    B0   0745             MOVW    R1,TTY$W_WB_SIZE(R2)        ; Set TWP size
         30    90   0749             MOVB    #DYN$C_TWP,-                ; Set TWP structure type
      0A A2         074B                     TTY$B_WB_TYPE(R2)           ;
         06    90   074D             MOVB    #IPL$_QUEUEAST,-            ; Set the TWP fork IPL (for later use)
      0B A2         074F                     TTY$B_WB_FIPL(R2)           ;
10 A2    01    D0   0751             MOVL    #1,TTY$L_WB_FR3(R2)         ; Request refresh of read prompt
      30 A2    9E   0755             MOVAB   TTY$L_WB_DATA(R2),-         ; Set address of message start
      1C A2         0758                     TTY$L_WB_NEXT(R2)           ;
      04 AE    C1   075A             ADDL3   SAVED_R1(SP),-              ; Set address of message end
      1C A2         075D                     TTY$L_WB_NEXT(R2),-         ;
      20 A2         075F                     TTY$L_WB_END(R2)            ;
     96'AF     9E   0761             MOVAB   B^END_BROADCAST,-           ; Set callback address
      2C A2         0764                     TTY$L_WB_RETADDR(R2)        ;
      24 A2    D4   0766             CLRL    TTY$L_WB_IRP(R2)            ; Clear pointer to associated IRP
         52    DD   0769             PUSHL   R2                          ; Save TWP address
      08 AE    28   076B             MOVC3   4+SAVED_R1(SP),-            ; Copy the message text to the TWP
      0C BE         076E                     @4+SAVED_R2(SP),-           ; (note the stack depth changed)
      30 A2         0770                     TTY$L_WB_DATA(R2)           ;
                    0772             ;
                    0772             ; Queue the broadcast request to the terminal.
                    0772             ; The disposition of the broadcast request will be determined
                    0772             ; by the contents of TTY$L_WB_END.  Note that if the request is
                    0772             ; accepted by a remote terminal, or is rejected outright, the
                    0772             ; TWP is no longer needed, and may be deallocated.  The TTY$L_WB_END
                    0772             ; field of the TWP will contain one of the following values:
                    0772             ;
                    0772             ;       System address: request accepted by TTDRIVER
                    0772             ;       1:              request accepted by RTTDRIVER
                    0772             ;       2:              request rejected
                    0772             ;
   53    6E    D0   0772             MOVL    (SP),R3                     ; Put TWP address in R3
55    18 AE    D0   0775             MOVL    4+SAVED_R5(SP),R5           ; Restore UCB address
      F884'    30   0779             BSBW    EXE$ALTQUEPKT               ; Queue the request to the terminal
         50  8ED0   077C             POPL    R0                          ; Remove TWP address from the stack
   6E    01    3C   077F             MOVZWL  #SS$_NORMAL,SAVED_R0(SP); Assume success
      20 A0    D5   0782             TSTL    TTY$L_WB_END(R0)            ; Check for rejection
         05    13   0785             BEQL    69$                         ; Branch if request rejected
         08    14   0787             BGTR    80$                         ; Branch if remote terminal accepted
         3F    BA   0789     13$:    POPR    #^M<R0,R1,R2,R3,R4,R5>      ; Restore the registers
               05   078B     14$:    RSB                                 ; Return
    0084 8F    3C   078C     69$:    MOVZWL  #SS$_DEVOFFLINE,-           ; Set broadcast rejection status
         6E         0790                     SAVED_R0(SP)                ;
      F86C'    30   0791     80$:    BSBW    COM$DRVDEALMEM              ; Deallocate the TWP
         F3    11   0794             BRB     13$                         ; Take common exit path
                    0796
                    0796     ;
                    0796     ; The following code performs all of the necessary broadcast post-processing.
                    0796     ; This entry point is FORKed to at IPL IPL$_QUEUEAST from the terminal driver.
                    0796     ; The fork block is the TWP.
                    0796     ;
                    0796     END_BROADCAST:                              ; Post-processor for broadcast requests
50       55    D0   0796             MOVL    R5,R0                       ; Copy TWP address
      F864'    31   0799             BRW     EXE$DEANONPAGED             ; Deallocate the TWP and return
```

```
                         079C         .SBTTL  BROADCAST EMERGENCY MESSAGE TO CONSOLE
                         079C  ;++
                         079C  ; IOC$CONBRDCST
                         079C  ;
                         079C  ;       This routine will allow emergency messages to be put on the console
                         079C  ;       terminal.  Some time later the broadcast will complete, and
                         079C  ;       at that time all the necessary post-processing will be done.
                         079C  ;
                         079C  ; Input:
                         079C  ;
                         079C  ;       R1 = Message length
                         079C  ;       R2 = Message address
                         079C  ;
                         079C  ; Implicit input:
                         079C  ;
                         079C  ;       IPL$_ASTDEL  <=  CURRENT_IPL  <=  UCB$B_FIPL(R5)
                         079C  ;
                         079C  ;       A dedicated TWP block must immediately preced the message.
                         079C  ;       The low bit of the first byte of the TWP is assumed to remain clear
                         079C  ;       while it is in use.
                         079C  ;
                         079C  ; Output:
                         079C  ;
                         079C  ;       None.  The contents of R1 .. R5 are preserved across the call.
                         079C  ;
                         079C  ; Routine value:
                         079C  ;
                         079C  ;       SS$_NORMAL      - The broadcast completed successfully.
                         079C  ;--
                         079C
               00000000  079C  SAVED_R0 = 0                                 ;.
               00000004  079C  SAVED_R1 = 4                                 ; .
               00000008  079C  SAVED_R2 = 8                                 ; .
               0000000C  079C  SAVED_R3 = 12                                ; . Symbolic offsets to saved registers
               00000010  079C  SAVED_R4 = 16                                ; .
               00000014  079C  SAVED_R5 = 20                                ;.
                         079C
                         079C  IOC$CONBRDCST::                              ; Broadcast to a terminal
                3F   BB  079C          PUSHR   #^M<R0,R1,R2,R3,R4,R5>   ; Save R0 .. R5
   55   00000000'EF  9E  079E          MOVAB   OPA$UCB0,R5              ; Get the console terminal UCB
             52   30  C2  07A5          SUBL2   #TTY$K_WB_LENGTH,R2      ; Retreat to the start of the TWP
                         07A8          ;
                         07A8          ; Fill in the Terminal Write Packet (TWP).
                         07A8          ;
         08 A2   51   B0  07A8          MOVW    R1,TTY$W_WB_SIZE(R2)     ; Set TWP size
                 30   90  07AC          MOVB    #DYN$C_TWP,-             ; Set TWP structure type
              0A A2       07AE                  TTY$B_WB_TYPE(R2)        ;
                 06   90  07B0          MOVB    #IPL$_QUEUEAST,-         ; Set the TWP fork IPL (for later use)
              08 A2       07B2                  TTY$B_WB_FIPL(R2)        ;
         10 A2   01   D0  07B4          MOVL    #1,TTY$L_WB_FR3(R2)      ; Request refresh of read prompt
              30 A2   9E  07B8          MOVAB   TTY$L_WB_DATA(R2),-      ; Set address of message start
              1C A2       07BB                  TTY$L_WB_NEXT(R2)        ;
              04 AE   C1  07BD          ADDL3   SAVED_R1(SP),-           ; Set address of message end
              1C A2       07C0                  TTY$L_WB_NEXT(R2),-      ;
              20 A2       07C2                  TTY$L_WB_END(R2)         ;
              EC'AF   9E  07C4          MOVAB   B^END_CONBRDCST,-        ; Set callback address
              2C A2       07C7                  TTY$L_WB_RETADDR(R2)     ;
```

```
  24 A2     D4  07C9                 CLRL    TTY$L_WB_IRP(R2)           ; Clear pointer to associated IRP
     52     DD  07CC                 PUSHL   R2                         ; Save TWP address
                07CE                 ;
                07CE                 ; Queue the broadcast request to the terminal.
                07CE                 ;
53   52     D0  07CE                 MOVL    R2,R3                      ; Put TWP address in R3
   F82C'    30  07D1                 BSBW    EXE$ALTQUEPKT              ; Queue the request to the terminal
     50   8ED0  07D4                 POPL    R0                         ; Remove TWP address from the stack
6E   01     3C  07D7                 MOVZWL  #SS$_NORMAL,SAVED_R0(SP); Assume success
  20 A0     D5  07DA                 TSTL    TTY$L_WB_END(R0)           ; Check for rejection
     03     13  07DD                 BEQL    69$                        ; Branch if request rejected
     3F     BA  07DF         13$:    POPR    #^M<R0,R1,R2,R3,R4,R5>     ; Restore the registers
            05  07E1         14$:    RSB                                ; Return
0084 8F     3C  07E2         69$:    MOVZWL  #SS$_DEVOFFLINE,-          ; Set broadcast rejection status
     6E         07E6                         SAVED_R0(SP)               ;
60   01     CE  07E7         80$:    MNEGL   #1,(R0)                    ; Mark the TWP free
     F3     11  07EA                 BRB     13$                        ; Take common exit path
                07EC
                07EC         ;
                07EC         ; The following code performs all of the necessary broadcast post-processing.
                07EC         ; This entry point is FORKed to at IPL IPL$_QUEUEAST from the terminal driver.
                07EC         ; The fork block is the TWP.
                07EC         ;
                07EC         END_CONBRDCST:                             ; Post-processor for broadcast requests
65   01     CE  07EC                 MNEGL   #1,(R5)                    ; Mark the TWP free
            05  07EF                 RSB
```

```
                          07F0          .SBTTL  SCAN THE I/O DATA BASE
                          07F0  ;+
                          07F0  ; IOC$SCAN_IODB - Scan the I/O data base and return next block.
                          07F0  ;
                          07F0  ; This routine is called to scan the device lists in the IO data base and
                          07F0  ; return a pointer to the next block in the list.  Context is kept in R11
                          07F0  ; and by using back pointers.
                          07F0  ;
                          07F0  ; Inputs:
                          07F0  ;
                          07F0  ;       The I/O data base is locked for read access.  This means that the caller
                          07F0  ;       owns the I/O data base mutex and/or is at IPL SYNCH or higher.
                          07F0  ;
                          07F0  ;               R11 =  0 implies first call
                          07F0  ;               R11 <> 0 indicates that R11 is pointer to current DDB
                          07F0  ;               R10 =  0 implies end of UCB chain
                          07F0  ;               R10 <> 0 indicates that R10 is pointer to current UCB
                          07F0  ;
                          07F0  ; Outputs:
                          07F0  ;
                          07F0  ;               R0  = Success status.
                          07F0  ;               R10 = Pointer to UCB
                          07F0  ;               R11 = Pointer to DDB
                          07F0  ;
                          07F0  ;       All other registers preserved.
                          07F0  ;
                          07F0  ;-
                          07F0
                          07F0  IOC$SCAN_IODB::
                          07F0
            50    01  D0  07F0          MOVL    #1,R0                   ; Success
                  5B  D5  07F3          TSTL    R11                     ; Initial condition?
                  2C  13  07F5          BEQL    50$                     ; Yes
                  5A  D5  07F7          TSTL    R10                     ; End of chain?
                  07  13  07F9          BEQL    10$                     ; Yes
        5A    30  AA  D0  07FB          MOVL    UCB$L_LINK(R10),R10     ; Get next UCB
                  01  13  07FF          BEQL    10$                     ; None
                  05      0801          RSB
                          0802
                  6B  D5  0802  10$:    TSTL    DDB$L_LINK(R11)         ; At end of DDB chain?
                  0A  13  0804          BEQL    30$                     ; Yes
            5B    6B  D0  0806          MOVL    DDB$L_LINK(R11),R11     ; No, get next one
        5A    04  AB  D0  0809  20$:    MOVL    DDB$L_UCB(R11),R10      ; Pick up first UCB
                  F3  13  080D          BEQL    10$                     ; None, get next DDB
                  05      080F          RSB
                          0810
        5B    34  AB  D0  0810  30$:    MOVL    DDB$L_SB(R11),R11       ; Get back to parent system block
            5B    6B  D0  0814  40$:    MOVL    SB$L_FLINK(R11),R11     ; Get next system block
   00000000'8F    5B  D1  0817          CMPL    R11,#SCS$GQ_CONFIG      ; End of chain?
                  0A  12  081E          BNEQ    60$                     ; No
                  50  D7  0820          DECL    R0
                  05      0822          RSB
                          0823
   5B    00000000'9F  D0  0823  50$:    MOVL    @#SCS$GQ_CONFIG,R11     ; Pick up first system block
              54  AB  D5  082A  60$:    TSTL    SB$L_DDB(R11)           ; Is there a DDB chain?
                  E5  13  082D          BEQL    40$                     ; No, go try next SB
        5B    54  AB  D0  082F          MOVL    SB$L_DDB(R11),R11       ; Yes, get the first DDB
```

```
                            D4   11  0833  2188          BRB     20$
                                     0835  2189
```

```
                     0835               .SBTTL  SCAN THE I/O DATA BASE BOTH PRIMARY & SECONDARY PATHS
                     0835       ;++
                     0835       ; IOC$SCAN_IODB_2P
                     0835       ;
                     0835       ; This routine is called to scan the device lists in the IO data base and
                     0835       ; return a pointer to the next block in the list.  Context is kept in R10
                     0835       ; and R11 and by using back pointers.
                     0835       ;
                     0835       ; SCAN_IODB_2P differs from SCAN_IODB in that it will scan both the primary
                     0835       ; and secondary UCB chain for each DDB.  This means that if a device is
                     0835       ; dual-pathed, SCAN_IODB_2P will return the address of its UCB twice, once in
                     0835       ; the context of the primary controller and once in the context of the
                     0835       ; secondary.
                     0835       ;
                     0835       ; Inputs and Outputs are identical to IOC$SCAN_IODB.
                     0835       ;--
                     0835
                     0835       IOC$SCAN_IODB_2P::
                     0835
       50    01  D0  0835               MOVL    #1,R0                   ; Success
             5B  D5  0838               TSTL    R11                     ; Initial condition?
             41  13  083A               BEQL    60$                     ; Yes
             5A  D5  083C               TSTL    R10                     ; Caller signalled end of chain?
             1C  13  083E               BEQL    30$                     ; Yes, done with this DDB
                     0840       ;
                     0840       ; At this point we must decide if we're following the primary or secondary
                     0840       ; chain of UCBs on this DDB.
                     0840       ;
   5B     28 AA  D1  0840               CMPL    UCB$L_DDB(R10),R11      ; Are we traversing the primary chain?
             07  12  0844               BNEQ    10$                     ; Branch if we're following secondary
   5A     30 AA  D0  0846               MOVL    UCB$L_LINK(R10),R10     ; Get next UCB on primary chain
             09  13  084A               BEQL    20$                     ; Branch if none to try secondary chain
             05      084C               RSB                             ; Else return UCB address to caller
                     084D       ;
                     084D       ; Get next UCB on secondary chain.
                     084D       ;
5A      00A4 CA  D0  084D       10$:    MOVL    UCB$L_DP_LINK(R10),R10  ; Get next UCB on secondary chain
             08  13  0852               BEQL    30$                     ; Branch if none left
             05      0854               RSB                             ; Else return UCB address to caller
                     0855       ;
                     0855       ; No UCBs left on primary chain; traverse secondary chain if present.
                     0855       ;
   5A     40 AB  D0  0855       20$:    MOVL    DDB$L_DP_UCB(R11),R10   ; Get first UCB on secondary chain
             01  13  0859               BEQL    30$                     ; Branch if none to try next DDB
             05      085B               RSB                             ; Else return UCB address to caller
                     085C       ;
                     085C       ; Step to next DDB.
                     085C       ;
             6B  D5  085C       30$:    TSTL    DDB$L_LINK(R11)         ; At end of DDB chain?
             0A  13  085E               BEQL    40$                     ; Yes, try next system block
          5B 6B  D0  0860               MOVL    DDB$L_LINK(R11),R11     ; No, get next one
   5A     04 AB  D0  0863       35$:    MOVL    DDB$L_UCB(R11),R10      ; Pick up first UCB on primary chain
             EC  13  0867               BEQL    20$                     ; None, try for UCB on secondary chain
             05      0869               RSB                             ; Else return UCB address to caller
                     086A
                     086A       ;
                     086A       ; Step to next system block.
```

```
                           086A  ;
      5B    34 AB    D0    086A  40$:    MOVL    DDB$L_SB(R11),R11          ; Get back to parent system block
           5B  6B    D0    086E  50$:    MOVL    SB$L_FLINK(R11),R11        ; Get next system block
  00000000'8F  5B    D1    0871          CMPL    R11,#SCS$GQ_CONFIG         ; End of chain?
               0A    12    0878          BNEQ    70$                        ; No
               50    D7    087A          DECL    R0                         ; Signal end of IO scan
               05          087C          RSB
                           087D
5B    00000000'9F    D0    087D  60$:    MOVL    @#SCS$GQ_CONFIG,R11        ; Pick up first system block
            54 AB    D5    0881  70$:    TSTL    SB$L_DDB(R11)              ; Is there a DDB chain?
               E5    13    0887          BEQL    50$                        ; No, go try next SB
      5B    54 AB    D0    0889          MOVL    SB$L_DDB(R11),R11          ; Yes, get the first DDB
               D4    11    088D          BRB     35$                        ; Try for UCB on primary chain
```

```
                  088F          .SBTTL  IOC$CTRLINIT - Call driver controller init. routine
                  088F   ;++
                  088F   ; FUNCTIONAL DESCRIPTION:
                  088F   ;
                  088F   ;       For UNIBUS devices, the device CSR is tested for existance.  If this
                  088F   ;       test fails, a no routine call occurs and failure status is returned in
                  088F   ;       R0.  Input values for a device driver's controller initialization
                  088F   ;       routine are loaded into the proper registers, the routine starting
                  088F   ;       address is located, and if a routine exists, it is called.
                  088F   ;
                  088F   ; INPUTS:
                  088F   ;       R1      CSR address to use if IDB contains zero
                  088F   ;       R8      CRB address (primary)
                  088F   ;       R11     DDB address
                  088F   ;
                  088F   ; OUTPUTS:
                  088F   ;       R0      Status (success, or failure ==> UNIBUS CSR non-existant)
                  088F   ;       R1-R6   Destroyed
                  088F   ;--
                  088F
                  088F   ;++
                  088F   ; Controller initialization routine parameters:
                  088F   ;
                  088F   ; INPUTS:
                  088F   ;       R4      CSR address (for UNIBUS and MASSBUS devices)
                  088F   ;               SCSSYSTEMID address (for class drivers during SYSGEN driver
                  088F   ;                       loading)
                  088F   ;               zero for all others, including class drivers during power
                  088F   ;                       failure recovery
                  088F   ;       R5      IDB address (or zero if none exists)
                  088F   ;       R6      DDB address
                  088F   ;       R8      CRB address
                  088F   ;
                  088F   ; OUTPUTS:
                  088F   ;       Must preserve all registers except R0 through R4.
                  088F   ;
                  088F   ;--
                  088F
                  088F
                  088F   IOC$CTRLINIT::
                  088F
 55   2C A8   D0  088F          MOVL    CRB$L_INTD+VEC$L_IDB(R8), R5    ; Get IDB address.
         05   18  0893          BGEQ    10$                             ; Branch if none.
    54   65   D0  0895          MOVL    IDB$L_CSR(R5), R4               ; Get CSR address.
         05   19  0898          BLSS    20$                             ; Branch if really a CSR.
    54   51   D0  089A   10$:   MOVL    R1, R4                          ; Else, use supplied value,
         1B   11  089D          BRB     40$                             ; and skip CSR testing.
                  089F
 56   14 A5   D0  089F   20$:   MOVL    IDB$L_ADP(R5), R6               ; Get ADP address.
         15   18  08A3          BGEQ    40$                             ; If none, skip CSR test.
 0E A6   01   B1  08A5          CMPW    #AT$_UBA, ADP$W_ADPTYPE(R6)     ; Is this a UBA?
         0F   12  08A9          BNEQ    40$                             ; If not a UBA, skip CSR test.
    56   66   D0  08AB          MOVL    ADP$L_CSR(R6), R6               ; Get adapter config reg addr.
    50   54   D0  08AE          MOVL    R4, R0                          ; Setup CSR for test.
00000000'GF   16  08B1          JSB     G^EXE$TEST_CSR                  ; Test UNIBUS CSR.
    0E   50   E9  08B7          BLBC    R0, 90$                         ; Branch if no CSR present.
                  08BA
```

```
50    30 A8   D0  08BA   2319 40$:    MOVL    CRB$L_INTD+VEC$L_INITIAL(R8), R0 ; Get ctrl init rout addr.
         05   18  08BE   2320         BGEQ    80$                              ; Branch if none.
   56    5B   D0  08C0   2321         MOVL    R11, R6                          ; Get DDB address.
         60   16  08C3   2322         JSB     (R0)                             ; Call ctrl init routine.
                  08C5   2323
   50    01   D0  08C5   2324 80$:    MOVL    #1, R0                           ; Set success status.
              05  08C8   2325 90$:    RSB                                      ; Return w/ status.
```

```
                      08C9          .SBTTL  IOC$UNITINIT - Call driver unit init. routine
                      08C9   ;++
                      08C9   ; FUNCTIONAL DESCRIPTION:
                      08C9   ;
                      08C9   ;       Input values for a device driver's unit initialization routine are
                      08C9   ;       loaded into the proper registers, the routine starting address is
                      08C9   ;       located, and if a routine exists, it is called.
                      08C9   ;
                      08C9   ; INPUTS:
                      08C9   ;       R5      UCB address
                      08C9   ;       R8      CRB address (primary)
                      08C9   ;
                      08C9   ; OUTPUTS:
                      08C9   ;       R0-R4   Destroyed
                      08C9   ;
                      08C9   ; NOTES:
                      08C9   ;
                      08C9   ;       There are two unit initialization routine addresses in the I/O data
                      08C9   ;       base; CRB$L_INTD_VEC$L_UNITINIT and DDT$L_UNITINIT.  Normally, only
                      08C9   ;       one of these two places should contain a unit initialization routine
                      08C9   ;       address.  However, for the console block storage device, the both
                      08C9   ;       locations contain an address, and the DDT contains the address which
                      08C9   ;       must be used.
                      08C9   ;
                      08C9   ;       In this case, the CRB is shared by the console terminal and console
                      08C9   ;       block storage devices.  The CRB contains the address of the unit
                      08C9   ;       initialization routine for the console terminal, and the console
                      08C9   ;       terminal DDT contains no unit initialization routine address.  Thus
                      08C9   ;       the console terminal device "fits" the "normal" case.  However, the
                      08C9   ;       console block storage device DDT contains a unit initialization
                      08C9   ;       routine which differs from the console terminal unit initialization
                      08C9   ;       routine and whose address is stored in the DDT.
                      08C9   ;
                      08C9   ;       Since the CRB is shared and contains the wrong unit initialization
                      08C9   ;       routine address for the console block storage device, the DDT must be
                      08C9   ;       inspected first.  Initialization for the console terminal will be
                      08C9   ;       accomplished correctly regardless of which address is checked first.
                      08C9   ;
                      08C9   ;--
                      08C9
                      08C9   ;++
                      08C9   ; Unit initialization routine parameters:
                      08C9   ;
                      08C9   ; INPUTS:
                      08C9   ;       R3      CSR address (primary)
                      08C9   ;       R4      CSR address (secondary, same as primary if no secondary exists)
                      08C9   ;       R5      UCB address
                      08C9   ;
                      08C9   ; OUTPUTS:
                      08C9   ;       Must preserve all registers except R0 through R4.
                      08C9   ;
                      08C9   ;--
                      08C9
                      08C9
                      08C9   IOC$UNITINIT::
                      08C9
  50   0088 C5   D0   08C9          MOVL    UCB$L_DDT(R5), R0               ; Get DDT address.
```

```
           50    18 A0   D0  08CE             MOVL    DDT$L_UNITINIT(R0), R0              ; Get DDT unit init rout addr.
  000005E4'8F       50   D1  08D2             CMPL    R0, #IOC$RETURN                     ; Null unit init routine?
                    06   12  08D9             BNEQ    10$                                 ; Branch if real unit init rout.
           50    3C A8   D0  08DB             MOVL    CRB$L_INTD+VEC$L_UNITINIT(R8), R0   ; Get CRB unit init rout addr.
                    1A   18  08DF             BGEQ    90$                                 ; Branch if no unit init rout.
                             08E1
                    54   D4  08E1      10$:   CLRL    R4                                  ; Assume no IDB exists.
           53    2C A8   D0  08E3             MOVL    CRB$L_INTD+VEC$L_IDB(R8), R3        ; Get IDB address.
                    10   18  08E7             BGEQ    50$                                 ; Branch if none.
              53    63   D0  08E9             MOVL    IDB$L_CSR(R3), R3                   ; Get primary CSR.
              54    53   D0  08EC             MOVL    R3, R4                              ; Assume no sec. CRB exists.
           51    20 A8   D0  08EF             MOVL    CRB$L_LINK(R8), R1                  ; Get secondary CRB addr.
                    04   18  08F3             BGEQ    50$                                 ; Branch if none.
                             08F5             ASSUME  IDB$L_CSR EQ 0
           54    2C B1   D0  08F5             MOVL    @CRB$L_INTD+VEC$L_IDB(R1), R4       ; Get secondary CSR addr.
                             08F9
                    60   17  08F9      50$:   JMP     (R0)                                ; Call unit init routine, and
                             08FB                                                         ; return to caller.
                             08FB
                         05  08FB      90$:   RSB                                         ; No unit init routine to call:
                             08FC                                                         ; return to caller.
                         08FC         .SBTTL  Parse Device Name String
                         08FC
                         08FC  ;+
                         08FC  ;
                         08FC  ; IOC$PARSDEVNAM - parse device name string
                         08FC  ;
                         08FC  ; This routine parses a device name string, checking syntax and
                         08FC  ; extracting node name, allocation class number, and unit number.
                         08FC  ; If device type format is specified, it is converted into the internal
                         08FC  ; compressed format.
                         08FC  ;
                         08FC  ; INPUTS:
                         08FC  ;
                         08FC  ;       R8 = size of name string
                         08FC  ;       R9 = address of name string
                         08FC  ;       R10 = flags
                         08FC  ;
                         08FC  ; OUTPUTS:
                         08FC  ;
                         08FC  ;       R0 = SS$_NORMAL - valid name string
                         08FC  ;          = SS$_IVDEVNAM - invalid device name string
                         08FC  ;       R2 = unit number
                         08FC  ;       R3 = length of SCS node name at head of name string
                         08FC  ;            or allocation class number
                         08FC  ;            or device type code
                         08FC  ;       R8 = size of name string
                         08FC  ;       R9 = address of name string
                         08FC  ;       R10 = flags
                         08FC  ;       R4 - R7, R11 preserved
                         08FC  ;
                         08FC  ;-
                         08FC
                         08FC         .ENABLE LSB
                         08FC
                         08FC  IOC$PARSDEVNAM::
      0070 8F      BB    08FC         PUSHR   #^M<R4,R5,R6>          ; save working registers
             58    D5    0900         TSTL    R8                     ; check name string length
             28    13    0902         BEQL    30$                    ; branch if null - error
       54    58    7D    0904         MOVQ    R8,R4                  ; copy name string descriptor
 56    59    01    C3    0907         SUBL3   #1,R9,R6               ; default is no node no allocation
                         090B                                        ; class, set pointer before beginning
                         090B                                        ; of the string
 69    58    24    3A    090B         LOCC    #^A'$',R8,(R9)         ; scan name for a "$"
             03    13    090F         BEQL    10$                    ; failed to find one - no nodename
       56    51    D0    0911         MOVL    R1,R6                  ; found it, save pointer
             52    7C    0914  10$:   CLRQ    R2                     ; init unit number and node name
       50    65    9A    0916  20$:   MOVZBL  (R5),R0                ; get next character
    11 50    06    E1    0919         BBC     #6,R0,40$              ; br if code 0-^X3F - numeric or $
       50    20    8A    091D         BICB    #^X20,R0               ; collapse lower case to upper case
    5A 8F    50    91    0920         CMPB    R0,#^A'Z'              ; possible alphabetic?
             77    1A    0924         BGTRU   150$                   ; br if not
    41 8F    50    91    0926         CMPB    R0,#^A'A'              ; possible alphabetic?
             37    1E    092A         BGEQU   70$                    ; branch if OK - store it
             6F    11    092C  30$:   BRB     150$                   ; no - error
                         092E  ;
                         092E  ; Non alphabetic - may be numeric or "$"
                         092E  ;
```

```
          56   55   D1  092E  40$:    CMPL    R5,R6                   ; hit the "$" yet?
               0E   13  0931          BEQL    50$                     ; yes, deal with it
               54   1A  0933          BGTRU   80$                     ; past it, digits are unit number
          39   50   91  0935          CMPB    R0,#^A'9'               ; legal?
               63   1A  0938          BGTRU   150$                    ; no, error
          30   50   91  093A          CMPB    R0,#^A'0'               ; legal?
               24   1E  093D          BGEQU   70$                     ; yes, accept it as alpha
               5C   11  093F          BRB     150$                    ; no, error
                        0941    ;
                        0941    ; $ in device name - either node name or allocation class.
                        0941    ;
     53   55   59   C3  0941  50$:    SUBL3   R9,R5,R3                ; compute length of node name
               1C   12  0945          BNEQ    70$                     ; branch if non-null - process the $
                        0947    ;
                        0947    ; Process allocation class number.
                        0947    ;
               55   D6  0947  60$:    INCL    R5                      ; move over "$" to allocation
               54   D7  0949          DECL    R4                      ; class digits.
               6A   10  094B          BSBB    GETNUMBER               ; convert allocation class.
          53   52   D0  094D          MOVL    R2,R3                   ; store requested allocation class.
               4B   15  0950          BLEQ    150$                    ; leq zero is not legal.
          5A   04   88  0952          BISB    #IOC$M_CLASS,R10        ; set allocation class flag
          50   24   91  0955          CMPB    #^A'$',R0               ; was terminator a "$"?
               43   12  0958          BNEQ    150$                    ; if not, invalid device name.
          58   54   7D  095A          MOVQ    R4,R8                   ; reset device name - unit size.
               54   D5  095D          TSTL    R4                      ; check remaining string count
               B5   14  095F          BGTR    20$                     ; if characters remain, process them.
               3A   11  0961          BRB     150$                    ; else, invalid device name.
                        0963
          85   50   90  0963  70$:    MOVB    R0,(R5)+                ; store potentially upcased character
          AD   54   F5  0966          SOBGTR  R4,20$                  ; any more characters to scan?
                        0969    ;
                        0969    ; End of alpha scan. Make sure we actually got a non-null device name.
                        0969    ;
          58   54   C2  0969  80$:    SUBL    R4,R8                   ; subtract unit number from string
               2F   13  096C          BEQL    150$                    ; if eql no device name specified
               56   D6  096E          INCL    R6                      ; point past $ in node name
          55   56   D1  0970          CMPL    R6,R5                   ; see if we've processed any more chars
               09   1F  0973          BLSSU   90$                     ; branch if yes
          25   5A   E8  0975          BLBS    R10,150$                ; branch if physical - not valid
     21   5A   06   E1  0978          BBC     #IOC$V_ANY,R10,150$     ; or if not generic search for any
               0D   11  097C          BRB     100$                    ; node name only - verify end of string
                        097E    ;
                        097E    ; Process unit number and make sure there's no trailing junk.
                        097E    ;
               52   D4  097E  90$:    CLRL    R2                      ; init unit number to 0
               54   D5  0980          TSTL    R4                      ; see if there's anything left
               0B   15  0982          BLEQ    110$                    ; branch if not
          5A   01   88  0984          BISB    #IOC$M_PHY,R10          ; set physical flag
               2E   10  0987          BSBB    GETNUMBER               ; convert unit number
               54   D6  0989          INCL    R4                      ; return terminator to string count
               54   D5  098B  100$:   TSTL    R4                      ; reached end of string?
               0E   14  098D          BGTR    150$                    ; branch if not - error
     37   5A   01   E0  098F  110$:   BBS     #IOC$V_TYPE,R10,190$    ; branch if name is a device type
          50   01   D0  0993  120$:   MOVL    #SS$_NORMAL,R0          ; successful parse
        0070   8F   BA  0996  130$:   POPR    #^M<R4,R5,R6>           ; restore registers
                    05  099A          RSB                             ; and return
```

```
                              099B   ;
                              099B   ; Invalid device name
                              099B   ;
                    8E   D5   099B   140$:   TSTL     (SP)+                   ; pop GETNUMBER return address from stack
      50    0144    8F   3C   099D   150$:   MOVZWL   #SS$_IVDEVNAM,R0        ; set invalid device name
                    F2   11   09A2           BRB      130$
                              09A4
                              09A4   ;+
                              09A4   ; Routine to convert ASCII to integer
                              09A4   ;
                              09A4   ; Inputs:
                              09A4   ;
                              09A4   ;       R2      assumed zero
                              09A4   ;       R4      size of string
                              09A4   ;       R5      starting address of string
                              09A4   ;
                              09A4   ; Outputs:
                              09A4   ;
                              09A4   ;       R0      terminator character
                              09A4   ;       R2      converted number
                              09A4   ;       R4      size of string with number and terminator character removed
                              09A4   ;       R5      address of first character after number terminator character
                              09A4   ;-
                              09A4
               50   85   9A   09A4   160$:   MOVZBL   (R5)+,R0                ; get next character.
               50   30   82   09A7           SUBB     #^A'0',R0               ; base it at decimal digits.
                    10   1F   09AA           BLSSU    170$                    ; branch if not a decimal digit.
               09   50   91   09AC           CMPB     R0,#9                   ; is it a digit?
                    0B   1A   09AF           BGTRU    170$                    ; branch if not a decimal digit.
               52   0A   C4   09B1           MULL     #10,R2                  ; scale current unit number by 10
               52   50   C0   09B4           ADDL     R0,R2                   ; add new digit to accumulation.
                              09B7   GETNUMBER:
               EA   54   F4   09B7           SOBGEQ   R4,160$                 ; count off a character
                    04   11   09BA           BRB      180$                    ; branch if no more characters
                              09BC
         50   FF   A5    9A   09BC   170$:   MOVZBL   -1(R5),R0               ; restore terminator character.
 00008000   8F     52    D1   09C0   180$:   CMPL     R2,#32768               ; check number value
                    D2   1E   09C7           BGEQU    140$                    ; branch if not valid
                         05   09C9           RSB                              ; return to caller.
                              09CA
                              09CA   ;
                              09CA   ; Parse device type name. We do this last because all the regular device
                              09CA   ; name validation is necessary anyway. Now we just have to do the
                              09CA   ; additional checks and pack the characters.
                              09CA   ;
                    53   D5   09CA   190$:   TSTL     R3                      ; check if we saw node or alloc class
                    CF   12   09CC           BNEQ     150$                    ; branch if so - not valid
      50       55   59   C3   09CE           SUBL3    R9,R5,R0                ; compute total length of string
               50   58   C2   09D2           SUBL     R8,R0                   ; compute length of unit number string
               02   50   D1   09D5           CMPL     R0,#2                   ; must be two digits
                    C3   12   09D8           BNEQ     150$                    ; branch if not - not valid
               55   59   D0   09DA           MOVL     R9,R5                   ; copy name address again
               02   58   D1   09DD           CMPL     R8,#2                   ; check minimum name length
                    BB   1F   09E0           BLSSU    150$                    ; too short - out
   50   85   40   8F     83   09E2           SUBB3    #^A'A'-1,(R5)+,R0       ; get char and convert to compressed
53   05   11      50     F0   09E7           INSV     R0,#17,#5,R3            ; store compressed code
   50   85   40   8F     83   09EC           SUBB3    #^A'A'-1,(R5)+,R0       ; get char and convert to compressed
```

```
                53    05    0C     50   F0  09F1  2577         INSV    R0,#12,#5,R3            ; store compressed code
                            03     58   D1  09F6  2578         CMPL    R8,#3                   ; check how many chars left
                                   A2   1A  09F9  2579         BGTRU   150$                    ; string was longer than 5 - error
                                   0A   1F  09FB  2580         BLSSU   200$                    ; short - don't take 3rd alpha
                  50    85     40  8F   83  09FD  2581         SUBB3   #^A'A'-1,(R5)+,R0       ; get char and convert to compressed
                53    05    07     50   F0  0A02  2582         INSV    R0,#7,#5,R3             ; store compressed code
                            53     52   C0  0A07  2583  200$:  ADDL    R2,R3                   ; add in unit number
                            5A     01   8A  0A0A  2584         BICB    #IOC$M_PHY,R10          ; clear physical flag
                                 FF83   31  0A0D  2585         BRW     120$                    ; done
```

```
           0A10         .SBTTL  Search I/O Database for Device
           0A10
           0A10  ;+
           0A10  ;
           0A10  ; IOC$SEARCHINT - internal I/O database search
           0A10  ;
           0A10  ; This routine searches the I/O database for the specified device, using
           0A10  ; the specified search rules. Depending on the search, a lock may or may
           0A10  ; not be taken out on the device when it is found.
           0A10  ;
           0A10  ; Note! While this routine is non-paged and therefore may be called at
           0A10  ; elevated IPL, the device locking code it calls is not. Therefore, only
           0A10  ; searches with IOC$V_ANY may be called from elevated IPL.
           0A10  ;
           0A10  ; INPUTS:
           0A10  ;
           0A10  ;         R2 = unit number
           0A10  ;         R3 = length of SCS node name at head of name string
           0A10  ;              or allocation class number
           0A10  ;              or device type code
           0A10  ;         R8 = size of name string
           0A10  ;         R9 = address of name string
           0A10  ;         R10 = flags
           0A10  ;         R11 = address to store lock value block
           0A10  ;         I/O database mutex held, IPL 2
           0A10  ;
           0A10  ; OUTPUTS:
           0A10  ;
           0A10  ;         R0 = SS$_NORMAL - device found
           0A10  ;            = SS$_NOSUCHDEV - device not found
           0A10  ;            = SS$_NODEVAVL - device exists but not available according to rules
           0A10  ;            = SS$_DEVALLOC - device allocated to other user
           0A10  ;            = SS$_NOPRIV - failed device protection
           0A10  ;            = SS$_TEMPLATEDEV - can't allocate template device
           0A10  ;            = SS$_DEVMOUNT - device already mounted
           0A10  ;            = SS$_DEVOFFLINE - device marked offline
           0A10  ;         R5 = UCB
           0A10  ;         R6 = DDB
           0A10  ;         R7 = system block
           0A10  ;         R10 - R4, R8 - R11 preserved
           0A10  ;
           0A10  ;         Note: If failure, R5 - R7 point to the last structures looked at.
           0A10  ;
           0A10  ;-
           0A10
           0A10  ;
           0A10  ; Stack use:
           0A10  ;
00000000   0A10  SAVR2   = 0
00000004   0A10  SAVR3   = 4
00000008   0A10  SAVR4   = 8
0000000C   0A10  SAVR8   = 12
00000010   0A10  SAVR9   = 16
           0A10
           0A10
           0A10          .ENABLE LSB
           0A10
```

```
                              0A10  IOC$SEARCHINT::
          031C 8F     BB      0A10          PUSHR   #^M<R2,R3,R4,R8,R9>     ; save registers
                              0A14  ;
                              0A14  ; Search the system blocks for a suitable node. If we are doing a search
                              0A14  ; by allocation class, generic device type, or no node name is given,
                              0A14  ; all system blocks qualify.
                              0A14  ;
  57   00000000'EF    DE      0A14          MOVAL   SCS$GQ_CONFIG,R7        ; get head of SCS SB list
             50  67   D0      0A1B  10$:    MOVL    SB$L_F[INK(R7),R0       ; get next system block
   00000000'8F   50   D1      0A1E          CMPL    R0,#SCS$GQ_CONFIG       ; are we back at list head?
                 78   13      0A25          BEQL    50$                     ; branch if yes - all done
                              0A27
             57  50   D0      0A27          MOVL    R0,R7
        56    54 A7   DE      0A2A          MOVAL   SB$L_DDB-DDB$L_LINK(R7),R6 ; pick up DDB listhead
             55  56   D0      0A2E          MOVL    R6,R5                   ; make sure UCB is non-zero
                              0A31                                          ; if allocation class or generic dev,
           5A    06   93      0A31          BITB    #IOC$M_CLASS!IOC$M_TYPE,R10
                 27   12      0A34          BNEQ    30$                     ; check every system block
           58  0C AE  7D      0A36          MOVQ    SAVR8(SP),R8            ; get orig dev name descriptor
           53  04 AE  D0      0A3A          MOVL    SAVR3(SP),R3            ; get node name length
                 1D   13      0A3E          BEQL    30$                     ; branch if none - go ahead
        44 A7    53   91      0A40          CMPB    R3,SB$T_NODENAME(R7)    ; check node name length
                 D5   12      0A44          BNEQ    10$                     ; branch if not
  69    45 A7    53   29      0A46          CMPC3   R3,SB$T_NODENAME+1(R7),(R9) ; node names match?
                 CE   12      0A4B          BNEQ    10$                     ; branch if not
                              0A4D  ;
                              0A4D  ; Found a suitable system block. Search its DDB list.
                              0A4D  ;
           50    01   3C      0A4D  20$:    MOVZWL  #SS$_NORMAL,R0
  53    04 AE    01   C1      0A50          ADDL3   #1,SAVR3(SP),R3         ; include the "$"
           59    53   C0      0A55          ADDL    R3,R9                   ; skip over the nodename
           58    53   C2      0A58          SUBL    R3,R8                   ; adjust the length
                 52   15      0A5B          BLEQ    60$                     ; if no device name, just return SB
                              0A5D
           50    66   D0      0A5D  30$:    MOVL    DDB$L_LINK(R6),R0       ; get address of next DDB
                 5A   13      0A60          BEQL    80$                     ; if eql end of list
           56    50   D0      0A62          MOVL    R0,R6
        55    D4 A6   DE      0A65          MOVAL   <DDB$L_UCB-UCB$L_LINK>(R6),R5 ; initialize primary UCB address
           5A    20   8A      0A69          BICB    #IOC$M_2P,R10           ; new DDB - clear secondary flag
        5E 5A    01   E0      0A6C          BBS     #IOC$V_TYPE,R10,100$    ; branch if generic type search
        07 5A    02   E1      0A70          BBC     #IOC$V_CLASS,R10,40$    ; branch if no class to check
   3C A6    04 AE     D1      0A74          CMPL    SAVR3(SP),DDB$L_ALLOCLS(R6) ; else, is allo. class right?
                 E2   12      0A79          BNEQ    30$                     ; branch if not, try next DDB
  15 A6    69    58   29      0A7B  40$:    CMPC3   R8,(R9),DDB$T_NAME+1(R6) ; check device name
                 DB   12      0A80          BNEQ    30$                     ; if no match, try next DDB
        50    14 A6   9A      0A82          MOVZBL  DDB$T_NAME(R6),R0       ; get length of name in DDB
           50    58   D1      0A86          CMPL    R8,R0                   ; check name lengths
                 43   13      0A89          BEQL    100$                    ; if they match - OK
                 50   D7      0A8B          DECL    R0                      ; try subtracting out controller letter
           50    58   D1      0A8D          CMPL    R8,R0                   ; and see if this matches
                 CB   12      0A90          BNEQ    30$                     ; if not, keep trying
              39 5A   E9      0A92          BLBC    R10,100$                        ; branch if not physical search - OK
  41 8F    15 A640    91      0A95          CMPB    DDB$T_NAME+1(R6)[R0],#^A'A' ; is this controller A?
                 31   13      0A9B          BEQL    100$                    ; if so, search it
                 BE   11      0A9D          BRB     30$                     ; if not, keep looking
                              0A9F  ;
                              0A9F  ; End of search - no suitable device has been found
                              0A9F  ;
   50   0908 8F        3C     0A9F  50$:    MOVZWL  #SS$_NOSUCHDEV,R0          ; no device found
     4D 5A   04        E1     0AA4          BBC     #IOC$V_EXISTS,R10,140$     ; branch if not seen
   50   09B0 8F        3C     0AA8          MOVZWL  #SS$_NODEVAVL,R0           ; otherwise status is not available
             46        11     0AAD          BRB     140$
                              0AAF  ;
                              0AAF  ; To here if we're just returning a system block, with no device specified.
                              0AAF  ;
        56   66        D0     0AAF  60$:    MOVL    (R6),R6                    ; get first DDB
   55     04 A6        D0     0AB2          MOVL    DDB$L_UCB(R6),R5           ; and first UCB
             3D        11     0AB6          BRB     140$                       ; and return
                              0AB8  ;
                              0AB8  ; To here when all UCB's on a DDB have been searched.
                              0AB8  ;
      A1 5A  01        E0     0AB8  70$:    BBS     #IOC$V_TYPE,R10,30$        ; if generic type search, try next DDB
                              0ABC  ;
                              0ABC  ; To here when all DDB's on a system block have been searched.
                              0ABC  ;
        5A   06        93     0ABC  80$:    BITB    #IOC$M_CLASS!IOC$M_TYPE,R10 ; if generic type or alloc class
             0A        12     0ABF          BNEQ    90$                        ; keep searching system blocks
        5A   09        93     0AC1          BITB    #IOC$M_PHY!IOC$M_LOCAL,R10 ; if physical or local only
             D9        12     0AC4          BNEQ    50$                        ; we're done
          04 AE        D5     0AC6          TSTL    SAVR3(SP)                  ; if there was an explicit node
             D4        12     0AC9          BNEQ    50$                        ; we're done
           FF4D        31     0ACB  90$:    BRW     10$                        ; else go try next system block
                              0ACE  ;
                              0ACE  ; Found a suitable DDB. Search both its UCB lists for the right UCB.
                              0ACE  ;
        52   6E        7D     0ACE  100$:   MOVQ    SAVR2(SP),R2               ; get unit number and device type
54    00000000'EF      D0     0AD1          MOVL    SCH$GL_CURPCB,R4           ; get PCB address
                              0AD8  NEXTUCB:                                   ; re-entry for next UCB
      07 5A  05        E1     0AD8  110$:   BBC     #IOC$V_2P,R10,120$         ; branch if on primary path
   55   00A4 C5        D0     0ADC          MOVL    UCB$L_2P_LINK(R5),R5       ; link to next secondary unit.
             04        11     0AE1          BRB     130$
   55     30 A5        D0     0AE3  120$:   MOVL    UCB$L_LINK(R5),R5          ; link to next primary unit.
             11        13     0AE7  130$:   BEQL    150$                       ; branch if no more units.
             28        10     0AE9          BSBB    IOC$TESTUNIT               ; is this unit ok?
          07 50        E8     0AEB          BLBS    R0,140$                    ; branch if successful
      E6 5A  04        E1     0AEE          BBC     #IOC$V_EXISTS,R10,110$     ; keep going if we haven't seen it yet
          E3 5A        E9     0AF2          BLBC    R10,110$                   ; or if not physical search
        031C 8F        BA     0AF5  140$:   POPR    #^M<R2,R3,R4,R8,R9>        ; restore registers
                       05     0AF9          RSB                                ; and return
                              0AFA
      BA 5A  05        E2     0AFA  150$:   BBSS    #IOC$V_2P,R10,70$          ; branch if secondary path already searched
      55   9C A6       DE     0AFE          MOVAL   <DDB$L_2P_UCB -            ; initialize secondary UCB address.
                              0B02                  -UCB$L_2P_LINK>(R6),R5
             D4        11     0B02          BRB     110$                       ; go search secondary path
                              0B04
                              0B04          .DISABLE LSB
```

```
                     0B04          .SBTTL  Continue I/O Database Search
                     0B04
                     0B04   ;+
                     0B04   ;
                     0B04   ; IOC$SEARCHCONT - internal I/O database search
                     0B04   ;
                     0B04   ; This routine continues a search started with a call to IOC$SEARCHINT.
                     0B04   ; It uses IOC$SEARCHINT's outputs as the starting point at which to
                     0B04   ; resume.
                     0B04   ;
                     0B04   ; INPUTS:
                     0B04   ;
                     0B04   ;       R2 = unit number
                     0B04   ;       R3 = length of SCS node name at head of name string
                     0B04   ;            or allocation class number
                     0B04   ;            or device type code
                     0B04   ;       R5 = last UCB
                     0B04   ;       R6 = last DDB
                     0B04   ;       R7 = last system block
                     0B04   ;       R8 = size of name string
                     0B04   ;       R9 = address of name string
                     0B04   ;       R10 = flags
                     0B04   ;       R11 = address to store lock value block
                     0B04   ;       I/O database mutex held, IPL 2
                     0B04   ;
                     0B04   ; OUTPUTS:
                     0B04   ;
                     0B04   ;       R0 = SS$_NORMAL - device found
                     0B04   ;          = SS$_NOSUCHDEV - device not found
                     0B04   ;          = SS$_NODEVAVL - device exists but not available according to rules
                     0B04   ;          = SS$_DEVALLOC - device allocated to other user
                     0B04   ;          = SS$_NOPRIV - failed device protection
                     0B04   ;          = SS$_TEMPLATEDEV - can't allocate template device
                     0B04   ;          = SS$_DEVMOUNT - device already mounted
                     0B04   ;          = SS$_DEVOFFLINE - device marked offline
                     0B04   ;       R5 = UCB
                     0B04   ;       R6 = DDB
                     0B04   ;       R7 = system block
                     0B04   ;       R10 - R4, R8 - R11 preserved
                     0B04   ;
                     0B04   ;       Note: If failure, R5 - R7 point to the last structures looked at.
                     0B04   ;
                     0B04   ;-
                     0B04
                     0B04   IOC$SEARCHCONT::
      031C 8F   BB   0B04           PUSHR   #^M<R2,R3,R4,R8,R9>     ; save registers
   05 5A   08   E5   0B08           BBCC    #IOC$V_ALT,R10,10$      ; check if alternate UCB in use
55    00A8 C5   D0   0B0C           MOVL    UCB$L_DP_ALTUCB(R5),R5  ; link back to other to continue
           C5   11   0B11   10$:    BRB     NEXTUCB                 ; continue search
```

```
                          0B13   2801           .SBTTL  Check UCB Against Search Rules
                          0B13   2802
                          0B13   2803   ;+
                          0B13   2804   ;
                          0B13   2805   ; IOC$TESTUNIT - Check UCB Against Search Rules
                          0B13   2806   ;
                          0B13   2807   ; INPUTS:
                          0B13   2808   ;
                          0B13   2809   ;       R2 = unit number
                          0B13   2810   ;       R3 = device type code
                          0B13   2811   ;       R4 = PCB address
                          0B13   2812   ;       R5 = UCB address
                          0B13   2813   ;       R10 = flags
                          0B13   2814   ;       R11= address of lock value block
                          0B13   2815   ;
                          0B13   2816   ; OUTPUTS:
                          0B13   2817   ;
                          0B13   2818   ;       R0 = SS$_NORMAL - eligible for use according to flags
                          0B13   2819   ;          = SS$_NOSUCHDEV - wrong unit number
                          0B13   2820   ;          = SS$_DEVALLOC - device allocated to other user
                          0B13   2821   ;          = SS$_NOPRIV - failed device protection
                          0B13   2822   ;          = SS$_TEMPLATEDEV - can't allocate template device
                          0B13   2823   ;          = SS$_DEVMOUNT - device already mounted
                          0B13   2824   ;          = SS$_DEVOFFLINE - device marked offline
                          0B13   2825   ;
                          0B13   2826   ;-
                          0B13   2827
                          0B13   2828   IOC$TESTUNIT::
   50    0908 8F     3C   0B13   2829           MOVZWL  #SS$_NOSUCHDEV,R0          ; assume wrong device
           06 5A     E9   0B18   2830           BLBC    R10,10$                    ; branch if not physical search
      54 A5    52    B1   0B1B   2831           CMPW    R2,UCB$W_UNIT(R5)          ; is the unit number exactly right?
               56    12   0B1F   2832           BNEQ    70$                        ; branch to error if not right.
                          0B21   2833
      09 5A    01    E1   0B21   2834   10$:    BBC     #IOC$V_TYPE,R10,20$        ; branch if not searching for dev type
               00    ED   0B25   2835           CMPZV   #MSCP$V_MTYP_N,-
               16         0B27   2836                   #MSCP$V_MTYP_D1,-
   53    008C C5          0B28   2837                   UCB$L_MEDIA_ID(R5),R3      ; is this the requested type?
               49    12   0B2C   2838           BNEQ    70$                        ; branch if not
         5A    10    88   0B2E   2839   20$:    BISB    #IOC$M_EXISTS,R10          ; note eligible device seen
   0A 3C A5    03    E1   0B31   2840           BBC     #DEV$V_CDP,UCB$L_DEVCHAR2(R5),30$ ; is this served path to a local d
   55    00A8 C5     D0   0B36   2841           MOVL    UCB$L_DP_ALTUCB(R5),R5     ; yes, get local path UCB address.
   5A    0100 8F     A8   0B3B   2842           BISW    #IOC$M_ACT,R10             ; note alternate UCB in use
      03 5A    06    E1   0B40   2843   30$:    BBC     #IOC$V_ANY,R10,40$         ; if SEARCHALL, finish with success.
             0091    31   0B44   2844           BRW     150$
                          0B47   2845   ;
                          0B47   2846   ; Check the device reference count and allocation status.
                          0B47   2847   ;
   50    006C 8F     3C   0B47   2848   40$:    MOVZWL  #SS$_DEVMOUNT,R0           ; check if device is already mounted
   55 38 A5    13    E0   0B4C   2849           BBS     #DEV$V_MNT,UCB$L_DEVCHAR(R5),100$
   50    0840 8F     3C   0B51   2850           MOVZWL  #SS$_DEVALLOC,R0
   4B 64 A5    09    E0   0B56   2851           BBS     #UCB$V_MOUNTING,UCB$W_STS(R5),100$ ; branch if mount in progress
         5C A5       B5   0B5B   2852           TSTW    UCB$W_REFC(R5)             ; is reference count zero?
               19    13   0B5E   2853           BEQL    80$                        ; branch if reference count is zero.
      0B 5A    07    E1   0B60   2854           BBC     #IOC$V_MOUNT,R10,50$       ; if mounting...
      0A 5A    0A    E0   0B64   2855           BBS     #IOC$V_ALLOC,R10,60$       ; if shared mount
   0C 38 A5    17    E1   0B68   2856           BBC     #DEV$V_ALL,UCB$L_DEVCHAR(R5),80$ ; OK if not allocated
               03    11   0B6D   2857           BRB     60$                        ; otherwise check allocation
```

```
                         0B6F
           34 5A    E9   0B6F  50$:    BLBC    R10,100$                    ; allocate: error if not phy
60 A4      2C A5    D1   0B72  60$:    CMPL    UCB$L_PID(R5),PCB$L_PID(R4) ; does this process own the device?
              2D    12   0B77  70$:    BNEQ    100$                        ; branch to error if not our device.
                         0B79          ;
                         0B79          ; Check all the other miscellaneous junk that can make a device not
                         0B79          ; available.
                         0B79          ;
        50    24    3C   0B79  80$:    MOVZWL  #SS$_NOPRIV,R0              ; check if device is spooled
06 38 A5      06    E1   0B7C          BBC     #DEV$V_SPL,UCB$L_DEVCHAR(R5),90$ ; branch if not
                         0B81          IFNPRIV ALLSPOOL,100$,R4            ; else, process must have ALLSPOOL priv.
 50   0084 8F       3C   0B87  90$:    MOVZWL  #SS$_DEVOFFLINE,R0          ; check if device is available
15 38 A5      12    E1   0B8C          BBC     #DEV$V_AVL,UCB$L_DEVCHAR(R5),100$
10 64 A5      04    E1   0B91          BBC     #UCB$V_ONLINE,UCB$W_STS(R5),100$
 50   21DC 8F       3C   0B96          MOVZWL  #SS$_TEMPLATEDEV,R0         ; check if device is a template
06 64 A5      0D    E0   0B9B          BBS     #UCB$V_TEMPLATE,UCB$W_STS(R5),100$
           F45D'    30   0BA0          BSBW    EXE$CHKRDACCES              ; check device protection
           0A 50    E8   0BA3          BLBS    R0,120$                     ; continue if accessible
                         0BA6          ;
                         0BA6          ; To here on any error.
                         0BA6          ;
   05 5A      08    E5   0BA6  100$:   BBCC    #IOC$V_ALT,R10,110$         ; check if alternate UCB in use
 55   00A8 C5       D0   0BAA          MOVL    UCB$L_DP_ALTUCB(R5),R5      ; link back to other to continue
              05         0BAF  110$:   RSB                                 ; return
                         0BB0          ;
                         0BB0          ; We've passed all the local tests. Now try to take out the appropriate
                         0BB0          ; lock on the device.
                         0BB0          ;
        51    5B    D0   0BB0  120$:   MOVL    R11,R1                      ; value block address
              05    13   0BB3          BEQL    130$                        ; branch if none
              61    7C   0BB5          CLRQ    (R1)                        ; initialize value block
           08 A1    7C   0BB7          CLRQ    8(R1)
19 3C A5      00    E1   0BBA  130$:   BBC     #DEV$V_CLU,UCB$L_DEVCHAR2(R5),150$ ; br. if not cluster visible
      50      05    D0   0BBF          MOVL    #LCK$K_EXMODE,R0            ; assume exclusive lock
   0C 5A      0A    E0   0BC2          BBS     #IOC$V_ALLOC,R10,140$       ; branch if allocation requested
   08 5A      07    E1   0BC6          BBC     #IOC$V_MOUNT,R10,140$       ; branch if not mount mode
03 38 A5      17    E0   0BCA          BBS     #DEV$V_ALL,UCB$L_DEVCHAR(R5),140$ ; br. if allocated
      50      04    D0   0BCF          MOVL    #LCK$K_PWMODE,R0            ; mount, no allocation - use PW
           F42B'    30   0BD2  140$:   BSBW    IOC$LOCK_DEV                ; and try to take device lock
           CE 50    E9   0BD5          BLBC    R0,100$
      50      01    D0   0BD8  150$:   MOVL    #SS$_NORMAL,R0              ; indicate success
              05         0BDB          RSB
```

```
                             0BDC               .SBTTL  IOC$THREADCRB
                             0BDC
                             0BDC      ;++
                             0BDC      ;
                             0BDC      ; FUNCTIONAL DESCRIPTION:
                             0BDC      ;
                             0BDC      ;        This routine will thread a CRB onto the duetime chain headed by
                             0BDC      ;        IOC$CRBTMOUT.
                             0BDC      ;
                             0BDC      ; CALLING SEQUENCE:
                             0BDC      ;
                             0BDC      ;        JSB     IOC$THREADCRB
                             0BDC      ;
                             0BDC      ; INPUTS:
                             0BDC      ;
                             0BDC      ;        R3 -->  CRB
                             0BDC      ;
                             0BDC      ; OUTPUTS:
                             0BDC      ;
                             0BDC      ;        NONE
                             0BDC      ;
                             0BDC      ;--
                             0BDC
                             0BDC      IOC$THREADCRB::
                 50     DD   0BDC               PUSHL   R0                         ; Save a register
  50    00000000'GF     DE   0BDE               MOVAL   G^IOC$GL_CRBTMOUT, R0      ; Pointer to list head
                 60     D5   0BE5      10$:     TSTL    (R0)                       ; End of the line?
                 05     13   0BE7               BEQL    20$                        ; Yes, go add new one
           50    60     D0   0BE9               MOVL    (R0), R0                   ; No, get next block
                 F7     11   0BEC               BRB     10$                        ; Try, try again
                             0BEE
        60    14 A3     DE   0BEE      20$:     MOVAL   CRB$L_TIMELINK(R3),(R0)    ; Link the new block in
                 50   8ED0   0BF2               POPL    R0                         ; Restore register
                        05   0BF5               RSB                                ; Leave
                             0BF6
                             0BF6
                             0BF6               .END
```

```
$$BASE                           = 00000001                 DDB$L_SB                         = 00000034
$$DISPL                          = 00000008                 DDB$L_UCB                        = 00000004
$$GENSW                          = 00000001                 DDB$T_NAME                       = 00000014
$$HIGH                           = 00000007                 DDT$L_CANCEL                     = 0000000C
$$LIMIT                          = 00000006                 DDT$L_REGDUMP                    = 00000010
$$LOW                            = 00000001                 DDT$L_START                      = 00000000
$$MNSW                           = 00000001                 DDT$L_UNITINIT                   = 00000018
$$MXSW                           = 00000001                 DEALLOC_DESCRIP                    000004C6 R      02
ADD_DOLLAR                         000006CA R      02       DEV$M_MBX                        = 00100000
ADD_NODE                           000006C0 R      02       DEV$M_TRM                        = 00000004
ADP$C_NUMDATAP                   = 00000010                 DEV$V_2P                         = 00000004
ADP$L_CSR                        = 00000000                 DEV$V_ALL                        = 00000017
ADP$L_DPQBL                      = 00000018                 DEV$V_AVL                        = 00000012
ADP$L_DPQFL                      = 00000014                 DEV$V_CDP                        = 00000003
ADP$L_MRACTMDRS                  = 0000005C                 DEV$V_CLU                        = 00000000
ADP$L_MRQBL                      = 00000034                 DEV$V_FOD                        = 0000000E
ADP$L_MRQFL                      = 00000030                 DEV$V_MNT                        = 00000013
ADP$W_ADPTYPE                    = 0000000E                 DEV$V_NNM                        = 00000009
ADP$W_DPBITMAP                   = 00000060                 DEV$V_OPR                        = 00000007
ADP$W_MRFREGARY                  = 0000015E                 DEV$V_SPL                        = 00000006
ADP$W_MRNREGARY                  = 00000064                 DEV$V_TRM                        = 00000002
ALLOC_DESCRIP                      000004DF R      02       DIR...                           = 00000001
ALLOC_NAME                         0000068D R      02       DISKCHK                            00000198 R      02
AT$_UBA                          = 00000001                 DISPLAY_NAME                       000006AA R      02
BINNUM                             00000000                 DO_PMS                             000001B0 R      02
BOO$GL_SPTFREH                     ********   X    02       DYN$C_TWP                        = 00000030
BOO$GL_SPTFREL                     ********   X    02       DYN$C_UCB                        = 00000010
BUG$_INCONSTATE                    ********   X    02       EMB$B_DV_ERTCNT                  = 00000C10
BUG$_IVBYTEALGN                    ********   X    02       EMB$Q_DV_IOSB                    = 00000012
BUG$_UNSUPRTCPU                    ********   X    02       EMB$W_DV_STS                     = 0000001A
CAN$C_AMBXDGN                    = 00000002                 END_BROADCAST                      00000796 R      02
CAN$C_DASSGN                     = 00000001                 END_CONBRDCST                      000007EC R      02
CDRP$L_BCNT                      = FFFFFFD2                 ERL$RELEASEMB                      ********   X    02
CDRP$L_FPC                       = 0000000C                 EXDVNM                             000006E3 R      02
CDRP$L_FQFL                      = 00000000                 EXE$ALONONPAGED                    ********   X    02
CDRP$L_FR3                       = 00000010                 EXE$ALTQUEPKT                      ********   X    02
CDRP$L_FR4                       = 00000014                 EXE$CHKRDACCES                     ********   X    02
CDRP$L_IOQFL                     = FFFFFFA0                 EXE$DEANONPAGED                    ********   X    02
CDRP$L_RWCPTR                    = 00000028                 EXE$GB_CPUTYPE                     ********   X    02
CDRP$L_UBARSRCE                  = 0000003C                 EXE$GL_ABSTIM                      ********   X    02
CDRP$W_BOFF                      = FFFFFFD0                 EXE$GQ_SYSTIME                     ********   X    02
CLU$GL_CLUB                        ********   X    02       EXE$MOUNTVER                       ********   X    02
COM$DRVDEALMEM                     ********   X    02       EXE$TEST_CSR                       ********   X    02
COMMON_ALOUBAMAP                   0000036D R      02       FULL_NAME                          00000688 R      02
CRB$B_MASK                       = 0000000E                 GETNUMBER                          000009B7 R      02
CRB$L_INTD                       = 00000024                 IDB$L_ADP                        = 00000014
CRB$L_LINK                       = 00000020                 IDB$L_CSR                        = 00000000
CRB$L_TIMELINK                   = 00000014                 IDB$L_OWNER                      = 00000004
CRB$L_WQBL                       = 00000004                 IOC$ALLOSPT                        0000062B RG     02
CRB$L_WQFL                       = 00000000                 IOC$ALODATAP                       00000268 R      02
CRB$M_BSY                        = 00000001                 IOC$ALOMAPUDA                      0000031B R      02
CRB$V_BSY                        = 00000000                 IOC$ALOUBAMAP                      00000345 RG     02
DC$_DISK                         = 00000001                 IOC$ALOUBAMAPN                     0000033E RG     02
DDB$L_2P_UCB                     = 00000040                 IOC$ALOUBAMAPSP                    000003AF RG     02
DDB$L_ALLOCLS                    = 0000003C                 IOC$ALOUBMAPRM                     00000455 RG     02
DDB$L_DP_UCB                     = 00000040                 IOC$ALOUBMAPRMN                    0000044E RG     02
DDB$L_LINK                       = 00000000                 IOC$ALTREQCOM                      00000118 RG     02
```

```
IOC$BROADCAST                     0000072B RG      02      IOC$WFIRLCH                       00000607 RG      02
IOC$CANCELIO                      00000000 RG      02      IPL$_ASTDEL                     = 00000002
IOC$CONBRDCST                     0000079C RG      02      IPL$_IOPOST                     = 00000004
IOC$CREDIT_UCB                    ********     X   02      IPL$_QUEUEAST                   = 00000006
IOC$CTRLINIT                      0000088F RG      02      IRP$C_DIAGBUF                   = 0000004C
IOC$CVT_DEVNAM                    00000652 RG      02      IRP$L_IOQFL                     = 00000000
IOC$DALOCUBAMAP                   00000573 R       02      IRP$L_MEDIA                     = 00000038
IOC$DELETE_UCB                    ********     X   02      IRP$L_PID                       = 0000000C
IOC$DIAGBUFILL                    0000005B RG      02      IRP$L_SVAPTE                    = 0000002C
IOC$GL_CRBTMOUT                   ********     X   02      IRP$L_UCB                       = 0000001C
IOC$GL_PSBL                       ********     X   02      IRP$V_DIAGBUF                   = 00000007
IOC$INITIATE                      000001DB RG      02      IRP$W_CHAN                      = 00000028
IOC$LAST_CHAN                     00000020 RG      02      IRP$W_STS                       = 0000002A
IOC$LAST_CHAN_AMBX                00000017 RG      02      LCK$K_EXMODE                    = 00000005
IOC$LOCK_DEV                      ********     X   02      LCK$K_PWMODE                    = 00000004
IOC$MNTVER                        000001D2 RG      02      LOCAL_NAME                        000006CC R       02
IOC$M_2P                        = 00000020                 MMG$GL_SPTBASE                    ********     X   02
IOC$M_ALT                       = 00000100                 MNTVERPNDCHK                      000001B8 R       02
IOC$M_CLASS                     = 00000004                 MSCP$V_MTYP_D1                  = 00000016
IOC$M_EXISTS                    = 00000010                 MSCP$V_MTYP_N                   = 00000000
IOC$M_LOCAL                     = 00000008                 NEXTUCB                           00000AD8 R       02
IOC$M_PHY                       = 00000001                 NO_SECONDARY                      000006EB R       02
IOC$M_TYPE                      = 00000002                 NXTIRP                            00000189 R       02
IOC$PARSDEVNAM                    000008FC RG      02      OPA$UCB0                          ********     X   02
IOC$RELCHAN                       0000008A RG      02      PCB$L_PID                       = 00000060
IOC$RELDATAP                      00000293 RG      02      PCB$Q_PRIV                      = 00000084
IOC$RELDATAPUDA                   00000288 RG      02      PDT$L_ADP                       = 000000E0
IOC$RELMAPREG                     0000051A RG      02      PMS$END_IO                        ********     X   02
IOC$RELMAPUDA                     000004FF RG      02      PMS$GL_IOPFMPDB                   ********     X   02
IOC$RELSCHAN                      00000080 RG      02      PMS$START_IO                      ********     X   02
IOC$REQCOM                        00000143 RG      02      PMSEND                            0000017A R       02
IOC$REQDATAP                      00000208 RG      02      PR$_IPL                         = 00000012
IOC$REQDATAPNW                    0000021A RG      02      PR$_SID_TYP730                  = 00000003
IOC$REQDATAPUDA                   00000228 RG      02      PR$_SID_TYP750                  = 00000002
IOC$REQMAPREG                     00000309 RG      02      PR$_SID_TYP780                  = 00000001
IOC$REQMAPUDA                     000002F4 RG      02      PR$_SID_TYP790                  = 00000004
IOC$REQPCHANH                     000000E1 RG      02      PR$_SID_TYP8NN                  = 00000006
IOC$REQPCHANL                     000000EA RG      02      PR$_SID_TYP8SS                  = 00000005
IOC$REQSCHANH                     000000CD RG      02      PR$_SID_TYPUV1                  = 00000007
IOC$REQSCHANL                     000000D7 RG      02      PR$_SIRR                        = 00000014
IOC$RETURN                        000005E4 RG      02      PRV$V_ALLSPOOL                  = 00000004
IOC$SCAN_IODB                     000007F0 RG      02      PUTASCIC                          00000708 R       02
IOC$SCAN_IODB_2P                  00000835 RG      02      PUTCHAR                           00000719 R       02
IOC$SEARCHCONT                    00000B04 RG      02      PUTDOLLAR                         00000716 R       02
IOC$SEARCHINT                     00000A10 RG      02      PUTNUM                            000006F0 R       02
IOC$TESTUNIT                      00000B13 RG      02      REALLOC_CD_MAPREGS                00000561 R       02
IOC$THREADCRB                     00000BDC RG      02      RELDATAP_COMMON                   0000029F R       02
IOC$UNITINIT                      000008C9 RG      02      RELEASE                           00000195 R       02
IOC$V_2P                        = 00000005                 RESR0                             00000008
IOC$V_ALLOC                     = 0000000A                 RESR1                             0000000C
IOC$V_ALT                       = 00000008                 RESR2                             00000010
IOC$V_ANY                       = 00000006                 RESR3                             00000014
IOC$V_CLASS                     = 00000002                 RESR4                             00000018
IOC$V_EXISTS                    = 00000004                 SAVABS...                       = 0000001C
IOC$V_MOUNT                     = 00000007                 SAVED_R0                        = 00000000
IOC$V_TYPE                      = 00000001                 SAVED_R1                        = 00000004
IOC$WFIKPCH                       000005E5 RG      02      SAVED_R2                        = 00000008
```

```
SAVED_R3                            = 0000000C              UCB$L_FPC                       = 0000000C
SAVED_R4                            = 00000010              UCB$L_FQFL                      = 00000000
SAVED_R5                            = 00000014              UCB$L_FR3                       = 00000010
SAVR2                               = 00000000              UCB$L_IOQFL                     = 0000004C
SAVR3                               = 00000004              UCB$L_IRP                       = 00000058
SAVR4                               = 00000008              UCB$L_LINK                      = 00000030
SAVR8                               = 0000000C              UCB$L_MEDIA_ID                  = 0000008C
SAVR9                               = 00000010              UCB$L_OPCNT                     = 00000070
SB$L_DDB                            = 00000054              UCB$L_PID                       = 0000002C
SB$L_FLINK                          = 00000000              UCB$L_STS                       = 00000064
SB$T_NODENAME                       = 00000044              UCB$L_SVAPTE                    = 00000078
SCH$GL_CURPCB                         ********   X    02    UCB$M_BSY                       = 00000100
SCRLEN                                00000010              UCB$M_CANCEL                    = 00000008
SCS$GA_LOCALSB                        ********   X    02    UCB$M_INT                       = 00000002
SCS$GQ_CONFIG                         ********   X    02    UCB$M_TIM                       = 00000001
SCS$RESUMEWAITR                       ********   X    02    UCB$M_TIMOUT                    = 00000040
SECONDARY_NAME                        0000069A R      02    UCB$V_BSY                       = 00000008
SS$_BUFFEROVF                       = 00000601              UCB$V_DELETEUCB                 = 00000010
SS$_DEVALLOC                        = 00000840              UCB$V_ERLOGIP                   = 00000002
SS$_DEVMOUNT                        = 0000006C              UCB$V_MNTVERIP                  = 0000000E
SS$_DEVOFFLINE                      = 00000084              UCB$V_MNTVERPND                 = 00000013
SS$_ILLIOFUNC                       = 000000F4              UCB$V_MOUNTING                  = 00000009
SS$_INSFMEM                         = 00000124              UCB$V_ONLINE                    = 00000004
SS$_IVDEVNAM                        = 00000144              UCB$V_TEMPLATE                  = 0000000D
SS$_NODEVAVL                        = 000009B0              UCB$W_BCNT                      = 0000007E
SS$_NOPRIV                          = 00000024              UCB$W_BOFF                      = 0000007C
SS$_NORMAL                          = 00000001              UCB$W_REFC                      = 0000005C
SS$_NOSUCHDEV                       = 00000908              UCB$W_STS                       = 00000064
SS$_TEMPLATEDEV                     = 000021DC              UCB$W_UNIT                      = 00000054
TTY$B_WB_FIPL                       = 0000000B              VEC$B_DATAPATH                  = 00000013
TTY$B_WB_TYPE                       = 0000000A              VEC$B_NUMREG                    = 00000012
TTY$K_WB_LENGTH                     = 00000030              VEC$L_ADP                       = 00000014
TTY$L_WB_DATA                       = 00000030              VEC$L_IDB                       = 00000008
TTY$L_WB_END                        = 00000020              VEC$L_INITIAL                   = 0000000C
TTY$L_WB_FR3                        = 00000010              VEC$L_UNITINIT                  = 00000018
TTY$L_WB_IRP                        = 00000024              VEC$M_MAPLOCK                   = 00008000
TTY$L_WB_NEXT                       = 0000001C              VEC$S_DATAPATH                  = 00000005
TTY$L_WB_RETADDR                    = 0000002C              VEC$V_DATAPATH                  = 00000000
TTY$W_WB_SIZE                       = 00000008              VEC$V_MAPLOCK                   = 0000000F
UBMD$B_DATAPATH                     = 00000003              VEC$V_PATHLOCK                  = 00000007
UBMD$B_NUMREG                       = 00000002              VEC$W_MAPREG                    = 00000010
UBMD$W_MAPREG                       = 00000000
UCB$B_DEVCLASS                      = 00000040
UCB$B_ERTCNT                        = 00000080
UCB$B_FIPL                          = 0000000B
UCB$B_TYPE                          = 0000000A
UCB$L_2P_LINK                       = 000000A4
UCB$L_CRB                           = 00000024
UCB$L_DDB                           = 00000028
UCB$L_DDT                           = 00000088
UCB$L_DEVCHAR                       = 00000038
UCB$L_DEVCHAR2                      = 0000003C
UCB$L_DP_ALTUCB                     = 000000A8
UCB$L_DP_DDB                        = 000000A0
UCB$L_DP_LINK                       = 000000A4
UCB$L_DUETIM                        = 0000006C
UCB$L_EMB                           = 00000094
```

```
                                        +----------------+
                                        ! Psect synopsis !
                                        +----------------+

PSECT name                 Allocation          PSECT No.  Attributes
----------                 ----------          ---------  ----------
.  ABS  .                  00000000  (    0.)  00 (  0.)  NOPIC    USR    CON    ABS    LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
$ABS$                      0000001C  (   28.)  01 (  1.)  NOPIC    USR    CON    ABS    LCL NOSHR   EXE   RD     WRT NOVEC BYTE
WIONONPAGED                00000BF6  ( 3062.)  02 (  2.)  NOPIC    USR    CON    REL    LCL NOSHR   EXE   RD     WRT NOVEC BYTE

                                      +------------------------+
                                      ! Performance indicators !
                                      +------------------------+

Phase                     Page faults    CPU Time       Elapsed Time
-----                     -----------    --------       ------------
Initialization                     29    00:00:00.06    00:00:01.71
Command processing                106    00:00:00.55    00:00:04.30
Pass 1                            693    00:00:31.36    00:01:37.67
Symbol table sort                   0    00:00:04.39    00:00:11.34
Pass 2                            403    00:00:08.26    00:00:26.97
Symbol table output                 1    00:00:00.25    00:00:00.66
Psect synopsis output               0    00:00:00.01    00:00:00.02
Cross-reference output              0    00:00:00.00    00:00:00.00
Assembler run totals             1234    00:00:44.90    00:02:22.68

The working set limit was 2400 pages.
182054 bytes (356 pages) of virtual memory were used to buffer the intermediate code.
There were 150 pages of symbol table space allocated to hold 2771 non-local and 169 local symbols.
2937 source lines were read in Pass 1, producing 24 object records in Pass 2.
59 pages of virtual memory were used to define 55 macros.

                                     +--------------------------+
                                     ! Macro library statistics !
                                     +--------------------------+

Macro library name                           Macros defined
------------------                           --------------
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                           35
_$255$DUA28:[SYSLIB]STARLET.MLB;2                        12
TOTALS (all libraries)                                   47

3009 GETS were required to define 47 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:IOSUBNPAG/OBJ=OBJ$:IOSUBNPAG MSRC$:IOSUBNPAG/UPDATE=(ENH$:IOSUBNPAG)+EXECML$/LIB
```

0376 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY